INtime® Software User Guide

TenAsys Corporation
1600 NW Compton Drive, Suite 104
Beaverton, OR 97006
(503) 748-4720
FAX: (503) 748-4730
www.tenasys.com

31001-2
January 2003

January 2003
Copyright © 2003 by TenAsys Corporation.
All rights reserved.

INtime, iRMX, and TenAsys are registered trademarks of TenAsys Corporation.

† All other trademarks, registered trademarks, service marks, and trade names are the property of their respective owners.

Before you begin

This guide describes INtime® software, an extension for Windows XP, Windows XP Embedded, Windows 2000, Windows NT 4.0, and Windows NT 4.0 Embedded, that provides the tools you need to create and run real-time (RT) applications—robust, high-performance applications with predictable responses to external events.

This guide assumes that you know how to develop programs for Windows and understand RT system concepts.

In this guide, the term “Windows” means either Windows XP, Windows XP Embedded, Windows 2000, Windows NT 4.0, or Windows NT Embedded 4.0.

About this guide

Guide contents

This guide introduces you to INtime software: how it makes RT applications possible, and how to use the INtime development tools. Use this guide to get acquainted with INtime software, then refer to Help for detailed information about INtime components. For more information about accessing help, see *Where to get more information* later in this chapter.

For a quick start, read *Chapter 1, Overview* to introduce you to all the basic INtime software concepts and to learn where to find detailed information about INtime software, then read *Chapter 10, INtime application development*, to learn about developing RT applications using INtime software.

Part I: Introducing INtime software

This part introduces INtime software and explains how INtime software and Windows work together to create RT applications.

Chapter		Description
1	Overview	Describes how INtime software works together with Windows to create and run RT applications, and lists INtime software's features. It also tells you where to find detailed information about INtime software topics.
2	Understanding INtime software architecture	Explains how INtime's RT kernel works with Windows to provide RT functionality. It also lists and describes INtime components.
3	About INtime software's RT kernel	Describes the RT kernel and its objects, the basic building blocks that application programs manipulate.
4	About RT programming	Describes processes unique to RT programming.
5	Designing RT applications	Provides general guidelines for RT system design.

Part II: Using INtime software

This part explains how to start INtime software and how to use the INtime software development tools.

Chapter		Description
6	Installation	Explains how to install and uninstall INtime software.
7	Configuration	Describes how to configure INtime software.
8	Preparing an RT node	Explains how to set up an RT node to run INtime software.
9	Operation	Describes how to start and run INtime software.
10	INtime application development	Explains how to use the INtime development environment to create INtime applications.

Part III: Appendices

The appendices provide additional information about INtime software.

Appendix		Description
A	INtime software system calls	Lists and describes system calls that threads in the RT portion of INtime applications use to communicate with each other and with Windows threads. You can find detailed information, including syntax and parameter values, in Help.
B	INtime software components	Lists and describes INtime software program files.

Appendix	Description
C Adding INtime software to an XP Embedded configuration	Lists and describes how to add INtime components to a Windows XP Embedded development environment so that XP Embedded images can be produced that include these INtime components.
D Troubleshooting	Lists problems you may encounter while running INtime software, and explains how to avoid or resolve those problems.

Glossary

The glossary defines terms used to describe INtime software.

Notational conventions

This manual uses the following conventions:

- All numbers are decimal unless otherwise stated.
- Bit 0 is the low-order bit. If a bit is set to 1, the associated description is true unless otherwise stated.
- `Data structures and syntax strings appear in this font.`

Notes indicate important information about the product.

Cautions indicate situations that may result in damage to data or the hardware.

Tips indicate alternate techniques or procedures that you can use to save time or better understand the product.

Warnings indicate situations that may result in physical harm to you or the hardware.

Where to get more information

About INtime software

You can find out more about INtime software from these sources:

- **World Wide Web**: TenAsys maintains an active site on the World Wide Web. The site contains current information about the company and locations of sales offices, new and existing products, contacts for sales, service, and technical support information. You can also send e-mail to TenAsys using the web site:

  `www.tenasys.com`

  You can contact TenAsys by email:

  `info@tenasys.com`

You can contact TenAsys technical support by email:

support@tenasys.com

When sending e-mail for technical support, please include information about both the hardware and software, plus a detailed description of the problem, including how to reproduce it.

Requests for sales, service, and technical support information receive prompt response.

- **INtime Help**: Describes INtime software concepts and explains how to use INtime tools. Help includes all system calls, including their syntax which you can cut and paste directly into your code. To access Help, do one of these:
  - **Within Microsoft Developer Studio**: Select Help>Contents. INtime Help displays.
  - **Within source code**: Highlight a system call in your source code, then press F1. Help for that system call displays.

    Before you can access context-sensitive Help within source code, you must enable context-sensitive Help with the Microsoft Developer Studio. To enable context-sensitive Help:

    i. In Microsoft Developer Studio, select the Help menu's Use Extension Help option. This disables the Microsoft online books, and enables INtime Help.

    ii. To display the parameters for any INtime software call, click on the call and press F1.

    To re-enable the Microsoft online books, select the Use Extension Help option entry again.
  - **Readme file**: Lists features and issues that arose too late to include in other documentation.
  - **Other**: If you purchased your TenAsys product from a third-party vendor, you can contact that vendor for service and support.

About Windows

For more information about Windows operation and program development, see these documents:

- Documentation that came with Windows.
- Documentation that came with Microsoft Developer Studio.
- Microsoft Windows resource kits (Server and workstation versions available).

Contents

Part I **Introducing INtime software**

Chapter 1 **Overview**

How does INtime software work? ... 3
Running an INtime application in Windows ... 3
Communication between Windows and RT threads ... 4
Considerations for INtime applications running on a single PC ... 5
Developing an INtime application ... 6
Design considerations ... 6
Code development ... 6
Features ... 7
Development environment ... 7
Wizards ... 7
Libraries ... 8
Debuggers ... 9
Sample applications ... 9
Runtime environment ... 11
RT enhancements to Windows ... 11
Memory protection ... 11
Blue screen protection ... 12

Chapter 2 **Understanding INtime software architecture**

How INtime software and Windows work together to run RT applications ... 13
Topology terminology ... 15
INtime on a single PC ... 15
INtime distributed across multiple PCs ... 15
Transport mechanisms ... 17
About the OSEM ... 18
How the RT interface driver works ... 19
About remote NTX ... 21
About the modified Windows HAL ... 21
About thread scheduling ... 22
Priority-based scheduling ... 22
Execution state ... 22
Round-robin scheduling ... 24
Handling interrupts ... 24
Interrupt handler alone ... 25
Interrupt handler/thread combination ... 25
Managing time ... 26
Alarm events ... 26

Chapter 3 About INtime software's RT kernel
What does the RT kernel provide? ... 27
RT kernel objects ... 27
Threads ... 28
Processes ... 28
Virtual memory ... 29
Memory pools ... 30
Dynamic memory ... 30
Object directories ... 31
Exchange objects ... 32
Validation levels ... 32
Mailboxes ... 33
Semaphores ... 34
Regions ... 35
Ports ... 36
Services ... 36
Heaps ... 36

Chapter 4 About RT programming
Multi-threading ... 38
Preemptive, priority-based scheduling ... 39
Interrupt processing ... 40
Multi-programming ... 42
Inter-thread coordination and communication ... 43
Messages ... 44
Synchronization ... 45
Mutual exclusion ... 45
Memory pools and memory sharing ... 46
System calls ... 48
Real time shared libraries ... 48
Exception handling ... 49

Chapter 5 Designing RT applications
Define the application ... 51
Target environments ... 53
Methodology ... 53
A hypothetical system ... 54
Interrupt and event processing ... 55
Multi-tasking ... 55

Part II Using INtime software

Chapter 6 Installation
Install INtime software on a Windows system ... 59
Requirements ... 59
Before you begin ... 60
Running the Installation program ... 60
Installing hardware for use with the RT kernel ... 62

Chapter 7 Configuration
Configuring INtime software ... 65
Default configuration ... 65
Running the INtime Configuration Utility ... 66
Miscellaneous ... 67
RTIF.SYS driver ... 67
Interrupt resources ... 67
Configuring INtime applications ... 67
Configuring Windows for non-interactive operation ... 68
Verifying BIOS execution ... 68
Configuring Windows for non-interactive logon ... 69
Configuring INtime software services to execute automatically ... 70
Using the ItWrpsrv.exe Service wrapper ... 70
Configuring a Windows service to execute automatically ... 71
Configuring the INtime TCP/IP software ... 71
Before you begin ... 71
Hardware installation ... 71
Setting the TCP/IP configuration parameters ... 72
NIC driver configuration ... 72
Advanced configuration parameters ... 73
Chapter 8 Preparing an RT node
Requirements ... 75
Configuring an RT subsystem ... 76
Building remote node boot/installation floppies ... 77
Booting from the Boot floppy disk ... 78
Copy build files to the hard/flash disk ... 78
Special requirements for Flash disks ... 79
Loading a user application on an RT node at boot time ... 79
RtLoad.xxx file format and switches ... 80
First character ... 80
First space ... 80
Editing the RtLoad file in-line ... 82
Chapter 9 Operation
Starting the RT kernel and related components ... 83
INtime service dependencies ... 85
After you start INtime software ... 85
Chapter 10 INtime application development
Create a project ... 88
Develop Windows source code ... 88
Adding the INtime RT Client Browser to your INtime application ... 88
Develop RT source code ... 90
Running the INtime RT process wizard ... 90
Running the INtime RT process add-in wizard ... 92
Running the INtime RT device driver wizard ... 92
Running the INtime RSL wizard ... 93

Compile 94
Debug 96
Debugging tips 96
Performance monitor 96
Status messages 97
Prepare for release 98
Before you begin 98
Using Runtime220.msi 98
Sample INtime applications 100
EventMsg DLL Project 100
INtime API Sample 101
INtime Serial Driver Sample 101
INtime Graphical Jitter 101
Real-time Interrupt Sample 102
C and C++ Samples for Debugger 102
TCPIP>TCP-IP Server Sample 102
TCPIP>TCIP-IP Client Sample 103
Fault Handling (ntrobust) 103
Floating Point Exception Handling 103
RSL Example 103
NTX Sample (MsgBoxDemo) 104
INtime Windows STOP Detection sample (STOPmgr) 104
INtime USB Client sample 104

Part III Appendices

Appendix A INtime software system calls

System call types 108
NTX calls 109
Handle conversion 109
RT calls 110
High-level (validating) calls 110
Low-level (non-validating) calls 110
RT services 111
RT system calls 112
Exception handling 112
High-level calls 112
Interrupts 112
High-level calls 112
Mailboxes 113
NTX calls 113
High-level calls 113
Low-level calls 114
Memory 114
Object directories 116

Ports 117
Service support 118
Port object management 118
Message transmission 119
Processes 119
Regions 120
Scheduler 120
Semaphores 121
Status 122
System accounting 123
System data 123
TCP/IP calls 124
Threads 125
Time management 126
RT services 127
Registry calls 127
RT service calls 128
RT service handlers 129
PCI library calls 130

Appendix B INtime software components

Blue.exe (Windows crash program) 132
BlueCon.rta 132
Clk0Jitr.rta 133
EventMsg.dll 133
INtimeConfiguration.exe 133
INtime.hlp 133
Main Help files 134
Utility Help files 134
Embedded C++ Help files 135
INscope.exe 135
INtex.exe 136
INtime Performance Monitor (INtmPerf.* files) 136
INtime RT Client Browser 136
ItWrpSrv.exe (Service wrapper) 137
Jitter.exe 137
LdRta.exe (INtime RT Application Loader) 137
LoadRtk.exe (INtime Kernel Loader) 138
MFC*.dll files 138
netstat.rta 139
NTX header files 139
NTX import libraries 139
NTX DLLs 139
NtxRemote2.exe (INtime Remote Connection Manager) 139
OvwGuide.pdf 140
Pdsproxy.dll files 140
Ping.rta 140
Project files 140

RT node files 141
RT header files 142
RT interface libraries 142
RT Stack Services 142
RT USB Interface Drivers 143
RtClkSrv.exe (INtime Clock Synchronization Service) 143
RtDrvrW5.awx (RT Device Driver wizard) 143
RtELServ.exe (INtime Event Log Service) 144
RtIf.sys (RT Interface Driver) 144
RtIOCons.exe (INtime I/O console) 145
RtIOSrv.exe (INtime I/O Service) 145
RtkImage.dbg (RT kernel with debug capabilities) 146
RtkImage (RT kernel) 146
RtMgr.exe (RT Process Manager) 146
RtNdSrv.exe (INtime Node Detection Service) 147
RtPrcMgr.rta 147
RtProcW5.awx (RT Process wizard) 147
RtProcAddinW5.awx (RT Process Add-in wizard) 147
RtRegSrv.exe (INtime Registry Service) 148
RtRslWiz.awx (RT Shared Library wizard) 148
Soft-Scope auxiliary files 148
SSBug.exe (Soft-Scope record conversion) 148
SSWin32.exe (Soft-Scope RT debugger) 149
Stacksrv.exe (INtime Network Service) 149
StarveAlarm.exe (Windows Starvation Alarm) 150

Appendix C **Adding INtime software to an XP Embedded configuration** 151

Appendix D **Troubleshooting**

1. Do a quick check 154
2. Look for symptoms 154
Other resources 158
Soft-Scope hints & tips 158

Glossary 151

Index 163

Figures

Figure 1-1. Transport mechanism for NTX communication 4
Figure 1-2. Transferring control between Windows and INtime software's RT kernel 5
Figure 1-3. Creating INtime applications 7
Figure 2-1. How Windows threads and RT threads communicate with each other on an INtime node 13
Figure 2-2. How INtime operates on a single PC 15
Figure 2-3. How INtime runs between computers 16
Figure 2-4. How NTX communicates with RT nodes 17
Figure 2-5. Encapsulating Windows processes and threads into an RT thread 19
Figure 2-6. Execution state transitions for threads 23
Figure 2-7. Round-robin scheduling 24
Figure 2-8. Thread execution model 26
Figure 3-1. Processes in a process tree 29
Figure 3-2. Threads using their process's memory pool 30
Figure 3-3. Threads using the root process's object directory 31
Figure 3-4. Threads using an object mailbox 33
Figure 3-5. Threads using a semaphore for synchronization 34
Figure 4-1. Thread switching in a multithreading environment 38
Figure 4-2. Multithreading and preemptive, priority-based scheduling 39
Figure 4-3. Interrupt handler interrupting a thread 40
Figure 4-4. Multiprogramming 42
Figure 4-5. Resources in a process 43
Figure 4-6. Object-based solution for message passing 44
Figure 4-7. Threads that use a semaphore for synchronization 45
Figure 4-8. Multithreading and mutual exclusion 46
Figure 4-9. Dynamic memory allocation between threads 47
Figure 5-1. Typical development cycle for INtime applications 52
Figure 5-2. The hardware of the dialysis application system 54
Figure 6-1. Installing INtime software 61
Figure 9-1. INtime service dependency hierarchy 85
Figure 10-1. Developing an INtime application 87
Figure 10-2. Installing INtime Runtime software 99
Figure A-1. Converting NTXHANDLES to RTHANDLES 109
Figure D-1. Troubleshooting INtime software problems 153

Tables

Table 3-1. Exchange objects: validation levels 32
Table 9-1. INtime software's Windows services 83
Table D-1. Symptom table 154
Table D-2. Solution table 155

I Introducing INtime software

This part acquaints you with INtime software: its components, how they're put together, and how they work with Windows to run RT applications.

This part contains:

Chapter 1: Overview

Describes how INtime software works together with Windows to create and run RT applications, and lists INtime software's features. It also tells you where to find detailed information about INtime software topics.

Read this chapter first. It introduces you to all the basic INtime software concepts and tells you where to find detailed information.

Chapter 2: Understanding INtime software architecture

Explains how INtime's RT kernel works with Windows to provide RT functionality. It also lists and describes INtime components.

Chapter 3: About INtime software's RT kernel

Describes the RT kernel and its objects, the basic building blocks that application programs manipulate.

Chapter 4: About RT programming

Describes processes unique to RT programming.

Chapter 5: Designing RT applications

Provides general guidelines for RT system design.

1 Overview

INtime software extends Windows to provide the tools you need to create and run RT (real-time) applications. INtime software consists of:

- **Development environment**: tools you use to create RT applications that run in the INtime runtime environment under Windows.
- **Runtime environment**: additions to your Windows system that provide RT performance for INtime applications.

This chapter describes how INtime software works together with Windows to create and run INtime applications, and lists INtime software features. It also tells you where to find detailed information about INtime software.

How does INtime software work?

You install INtime software on a system that already runs Windows. Once installed, Windows and INtime software work together to provide deterministic, RT support for INtime applications.

INtime software works with the following versions of Windows:

- Windows XP
- Windows XP Embedded
- Windows 2000
- Windows NT 4.0
- Windows NT Embedded 4.0

Running an INtime application in Windows

For detailed information about how INtime software and Windows work together to run INtime applications, see *Chapter 2, Understanding INtime software architecture.*

An INtime application includes these components:

- **RT processes**: RT processes contain threads that typically handle time-critical I/O and control. RT threads preempt Windows threads.

- **Windows processes**: Windows processes contain threads that handle aspects other than time-critical I/O and control, including the user interface, network communication, data manipulation and computation, and data storage.

Communication between Windows and RT threads

When an INtime application runs, Windows threads communicate with RT threads via the Windows extension (NTX) API.

The RT threads in your INtime application(s) may reside on the same PC as the Windows threads or in a remote computer accessed via serial or Ethernet cable. The NTX API automatically detects the connection type and determines the transport mechanism to use between Windows and RT threads: in-memory (local), serial, or Ethernet:

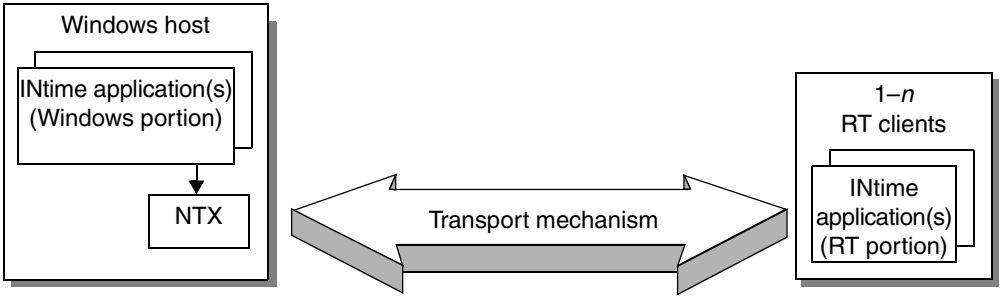
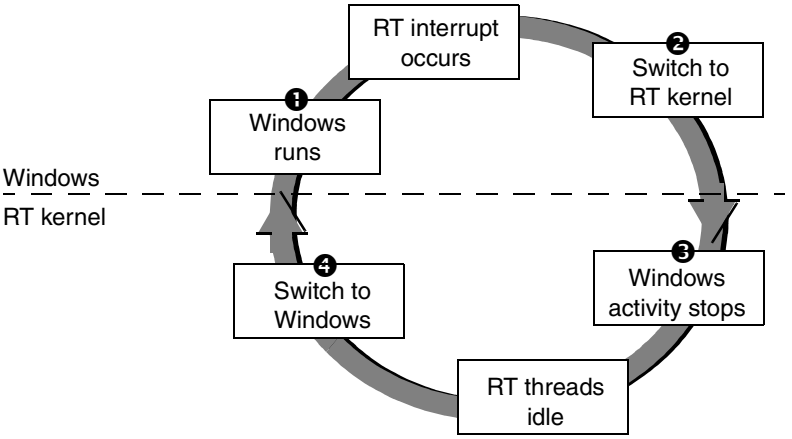


Figure 1-1. Transport mechanism for NTX communication

For detailed information about INtime software's transport mechanisms, see *Transport mechanisms* in *Chapter 2, Understanding INtime software architecture*

Considerations for INtime applications running on a single PC

When both the Windows and RT portions of an INtime application run on a single PC, INtime software transfers control between the Windows and RT environments as shown in this figure:



❶ When a Windows thread runs, the full Windows environment exists, including its interrupts, interrupt masks, and handlers.

❷ When an RT interrupt occurs, control immediately switches to the RT kernel, where an RT interrupt handler deals with the event. This, in turn, may cause one or more RT threads to execute.

❸ Windows processes and interrupts stop until the RT threads complete.

❹ When all RT threads complete their work, leaving no RT threads ready to run, control switches back to the Windows environment, and standard Windows scheduling resumes.

Figure 1-2. Transferring control between Windows and INtime software's RT kernel

When running on a single microprocessor, the INtime runtime environment encapsulates all Windows processes and threads into a single RT thread of lowest priority. As a result, RT threads always preempt running Windows threads, guaranteeing determinism for RT activities within the system.

The RT and Windows threads can share sections of memory allocated by INtime applications. A Windows thread can obtain a handle for this shared memory, then map the memory referenced by that handle into the thread's address space.

For detailed information about memory usage, go to Help and select About INtime software, RT kernel objects, then Memory Management.

Developing an INtime application

Design considerations

For detailed information about designing INtime applications, see *Chapter 5, Designing RT applications*.

When designing INtime applications, you must divide the labor appropriately between Windows processes and RT processes and, to a finer degree, between the threads in each process. For the best performance, limit RT processes to performing only time-critical functions, and determine which Windows threads require the greater relative priority.

Code development

For detailed information about developing INtime applications, see *Chapter 10, INtime application development*.

To develop an INtime application, you use Microsoft Developer Studio, including INtime wizards and Microsoft Developer Studio extensions for RT processes, a standard Windows debugger, and a Windows-based RT dynamic debugger that supports on-target debugging of RT threads.

When developing INtime applications with Microsoft Developer Studio, you create executable files for both RT and Windows environments as shown in this figure:

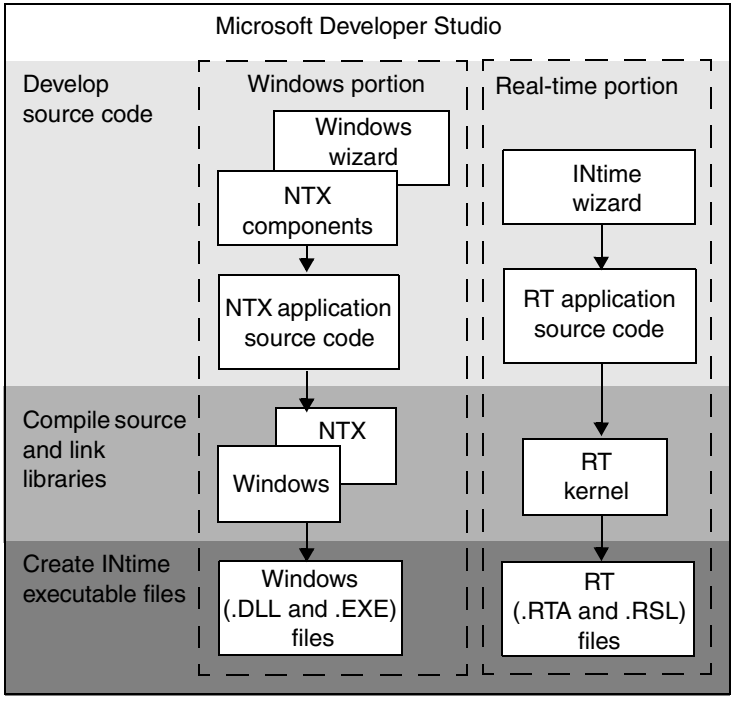


Figure 1-3. Creating INtime applications

Features

For detailed list of INtime software components, see *Appendix B, INtime software components*. For information about using these components, see *Chapter 10, INtime application development*.

Development environment

To develop INtime applications, you use standard Windows tools together with these INtime software tools:

Wizards

Accessed from within Microsoft Developer Studio, INtime wizards automatically prompt you for the information needed to create projects that contain source code for the RT portion of your INtime applications. Once you create the project, you manually edit the code.

INtime software provides these wizards:

- **RT Process wizard**: develops the RT portion of INtime applications.
- **RT Process Add-in wizard**: adds supplemental files to the already-generated RT portion of INtime applications.
- **RT Device Driver wizard**: develops RT device drivers.
- **RT Shared Library wizard**: develops RT shared library (RSL-RT equivalent to Windows DLL)

For information about using the INtime wizards, see *Chapter 10, INtime application development*.

Libraries

INtime software provides interface libraries that your threads use to obtain RT kernel services. These libraries support mailboxes, semaphores, and shared memory. INtime software libraries include:

- **Windows extension (NTX) library**: Contains system calls that the Windows portion of an INtime application uses to communicate with the RT portion of the system.
- **Real-time (RT) application library**: Contains system calls that the RT portion of an INtime application uses to access RT kernel services such as memory management and inter-thread communication.
- **Real-time (RT) DSM library**: Contains system calls that implement sponsorship and dependency registration of INtime RTAs (real-time applications) with their counter-part Windows applications.
- **Real-time C and EC++ libraries**: Contains system calls that the RT portion of an INtime application uses to access standard ANSI C and EC++ functions.
- **RT services library**: Contains additional calls required to implement INtime Services.
- **PCI library**: Contains system calls that provide access to the PCI bus configuration space.

For an overview of system calls included in the APIs, see *Appendix A, INtime software system calls*. For detailed information, including syntax and parameter values, see Help.

Debuggers

You debug the Windows portion of INtime applications using the debug tools provided in Microsoft Developer Studio. To debug the RT portion of INtime applications, you use the debug tools provided with INtime software:

- **Soft-Scope† (SSWIN32.EXE)**: A Windows application that provides source level, multi-tasking debug capabilities. Soft-Scope can debug multiple RT threads simultaneously while other threads continue to run.
- **System debug monitor (SDM)**: A command-line interface that provides low-level, static debugging capability. SDM requires a separate serial debug terminal for its user interface.
- **System Debugger (SDB)**: An RT subsystem that provides a mechanism to retrieve information about RT kernel objects, threads, and processes. You can activate this mechanism either from SDM in a static debug session, or from Soft-Scope in a dynamic debug session.

You can debug the Windows and RT portions of an INtime application simultaneously.

For detailed information about Soft-Scope, see Soft-Scope's Help. For detailed information about SDM, see INtime Help. For detailed information about using SDM, access INtime Help, then select Debuggers>Low-level debugger>System Debug Monitor (SDM).

Sample applications

INtime software contains several sample applications that you can use as examples for your own program development.

- **EventMsg DLL Project**: This DLL allows you to customize event messages.
- **INtime API Sample**:This test application exercises a subset of the INtime software.
- **INtime Serial Driver Sample**: This demo provides a high-speed serial driver in both binary and source form, plus an RT application that uses this driver to communicate with a terminal.
- **INtime Graphical Jitter**: This application measures the minimum, maximum, and average times between consecutive INtime low-level clock ticks via an Alarm Event Handler. Because this application consists of both RT and Windows executables, it shows both INtime threads and Windows NTX API usage.
- **Real-time Interrupt Sample**: This application tests the INtime RT Interrupt system calls using the Transmitter Ready interrupt from COM1.

- **C and C++ Samples for Debugger**: These simple C and C++ programs are provided as a vehicle to demonstrate the Soft-Scope debugger's capabilities. The C++ program also demonstrates several components of the C++ language available to RT applications, as well as basic classes, dynamic instantiation, operator overloading, and so on. It also shows the libraries and startup modules needed.
- **TCPIP>TCP-IP Server Sample**: Sample TCP/IP application that waits for 130Kbyte messages and returns them to the sender.
- **TCPIP>TCIP-IP Client Sample**: Sample TCP/IP application that looks for a TCP/IP server and then sends 130KByte messages to it and waits for the reply.
- **Fault Handling (ntrobust)**: This INtime application has both a Windows and a RT portion. The Windows portion allows the user to set up timing parameters that control how often a thread in the RT portion causes a hardware fault. The application demonstrates how another RT thread can detect and log the failure, delete the offending thread, and recreate it, all without affecting Windows or other RT processes.
- **Floating Point Exception Handling**: This sample program demonstrates floating point exception handling.
- **RSL Example**: This RT program demonstrates the creation and use of RT Shared Libraries, the RT analog for Windows DLLs.
- **NTX Sample (MsgBoxDemo)**: This INtime application has both a Windows and a RT portion. The Windows portion looks up an RT mailbox created by the RT portion, and then waits at the mailbox. When an RT thread sends a message to the mailbox, the Windows portion displays the received data in a message box on the Windows side. RT semaphore and RT shared memory usage are also demonstrated.
- **INtime Windows STOP Detection sample (STOPmgr)**: This sample application shows how an INtime application can detect either a Windows Crash (blue screen) or Windows Shutdown event and prevent Windows from completing its normal actions until the RT application has had a chance to do a "graceful" shutdown.

For detailed information about these sample applications, see *Chapter 10, INtime application development*.

Runtime environment

INtime's runtime environment includes RT enhancements to Windows, memory protection, and blue screen protection. INtime software uses these features only for INtime applications that run on a single microprocessor, unless otherwise noted. Runtime features are described in detail in the following sections.

RT enhancements to Windows

These features enable Windows and the RT kernel to work together:

- **RT kernel**: (used in both INtime and RT nodes) provides deterministic scheduling and execution of RT threads.
- **OS encapsulation mechanism (OSEM)**: (used only with INtime nodes) manages the simultaneous operation and integrity of the Windows kernel and the RT kernel.
- **Modified (at run-time) Windows Hardware Abstraction Layer (HAL)**: (required only for INtime nodes) ensures the determinism of INtime applications.

For information about the RT kernel, see *Chapter 3, About INtime software's RT kernel*. For information about the OSEM and HAL, see *Chapter 2, Understanding INtime software architecture*

Memory protection

During INtime node system initialization, memory is allocated through the OSEM for use by the RT kernel and INtime applications. This memory is "locked down" so that it does not page to disk. It is also removed from the memory pool available for Windows applications.

INtime's RT kernel provides several protection levels for RT memory:

- **32-bit segmentation**: (used in both INtime and RT nodes)
  - **Local configurations**: INtime software keeps Windows and each RT process in separate segments. Keeping Windows separate from the RT kernel isolates and protects addresses not only between complex RT processes but between RT processes and Windows processes.
  - **Remote configurations**: INtime software keeps RT processes in separate segments. Keeping them separate from the RT kernel isolates and protects addresses between complex RT processes.

- **Paging**: (used in both INtime and RT nodes) The RT kernel uses the processor's paging mode for virtual address translation, but does not implement demand paging. Each RT process is loaded into its own virtual address space, defined by a 32-bit virtual segment. Because code, data, and stack are automatically placed in non-contiguous areas of the application's virtual memory, memory overruns are trapped as page faults.
- **Virtual addressing**: (used in both INtime and RT nodes) Since each RT process resides in a separate memory space defined by a virtual segment created by the RT Application Loader, RT processes cannot address beyond the virtual segment. This effectively partitions every RT process into its own address space.

Blue screen protection

On an INtime node, the RT kernel enables successful execution of RT threads even in the event of a total Windows failure, also known as a "blue screen crash."

- **Failure diversion**: HAL modifications capture Windows failures. Once captured, control transfers to the RT kernel, Windows operation is suspended, and RT threads continue to run, unaffected by the failure.
- **Application-specific recovery**: (used only in local INtime system configurations) In the event of a Windows failure, your crash recovery RT threads run and you can execute an orderly shutdown of the hardware your INtime application controls.

Remote INtime configurations do not require blue-screen protection because RT processes execute on dedicated hardware, separately from Windows. This means a crash, reboot, or restart of the Windows system does not interrupt execution of remote RT processes.

In the event of a Windows blue screen crash, INtime software keeps running until a graceful shutdown occurs. To start INtime software again, you must first restart Windows.

2 Understanding INtime software architecture

This chapter explains how the RT kernel works with Windows to provide real-time functionality. It also lists and describes INtime components.

How INtime software and Windows work together to run RT applications

When an INtime application runs on an INtime node, Windows threads communicate with RT threads via the Windows extension (NTX) library as shown in this figure:

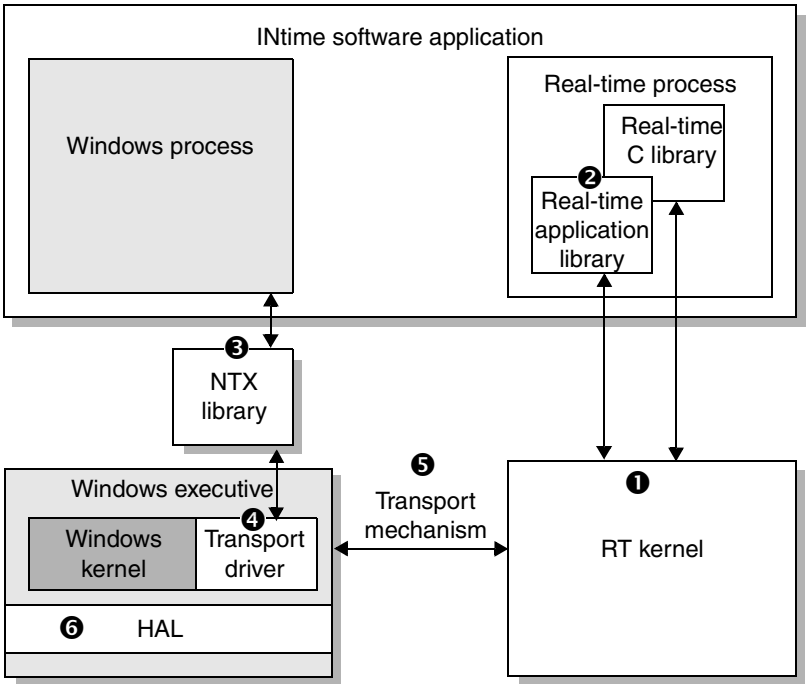


Figure 2-1. How Windows threads and RT threads communicate with each other on an INtime node

The INtime components include:

❶ **RT kernel**: Provides deterministic scheduling and execution of RT threads within RT processes. For detailed information about the kernel, see *Chapter 3, About INtime software's RT kernel*.

❷ **Real-time application, C, and EC++ libraries**: Gives direct access to the RT kernel services for RT threads. For an overview of calls in the RT libraries, see *Appendix A, INtime software system calls*. For detailed information on all calls, including syntax and parameter values, see Help.

❸ **NTX library**: Provides RT interface extensions for the Win32 API that allow Windows threads to communicate and exchange data with RT threads within the application. For an overview of calls in this library, see *Appendix A, INtime software system calls*. For detailed information, including syntax and parameter values, see Help.

❹ **Transport driver**: A driver that converts information to the protocol needed by the specified transport mechanism. For details, see *Transport mechanisms* later in this chapter.

❺ **Transport mechanism**: The communication protocol or method used by NTX to communicate between Windows and RT threads. Whether the various portions of your INtime applications reside on a single PC, or on multiple computers accessed via serial or Ethernet cable, NTX provides this essential communication. For details, see *Transport mechanisms* later in this chapter.

❻ **Windows hardware abstraction layer (HAL)**: INtime software intercepts some HAL calls to ensure real-time performance. For details, see *About the modified Windows HAL"*later in this chapter.

Topology terminology

INtime on a single PC

When INtime software runs as an INtime node, the Windows host communicates with the RT client via the OSEM.

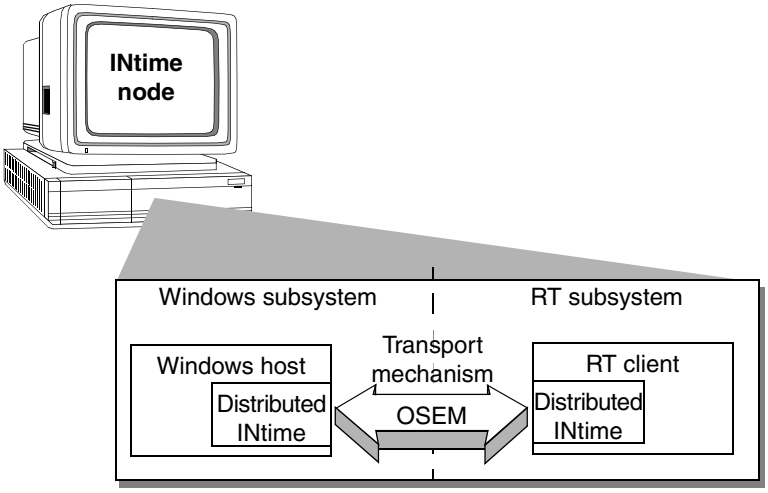


Figure 2-2. How INtime operates on a single PC

INtime distributed across multiple PCs

When INtime software runs between computers, the Windows host communicates with the RT client via RS-232 or Ethernet.

The Windows node is a computer that requires a Windows subsystem which contains the Windows host software.

The RT node is a computer that requires an RT subsystem which contains the RT client software.

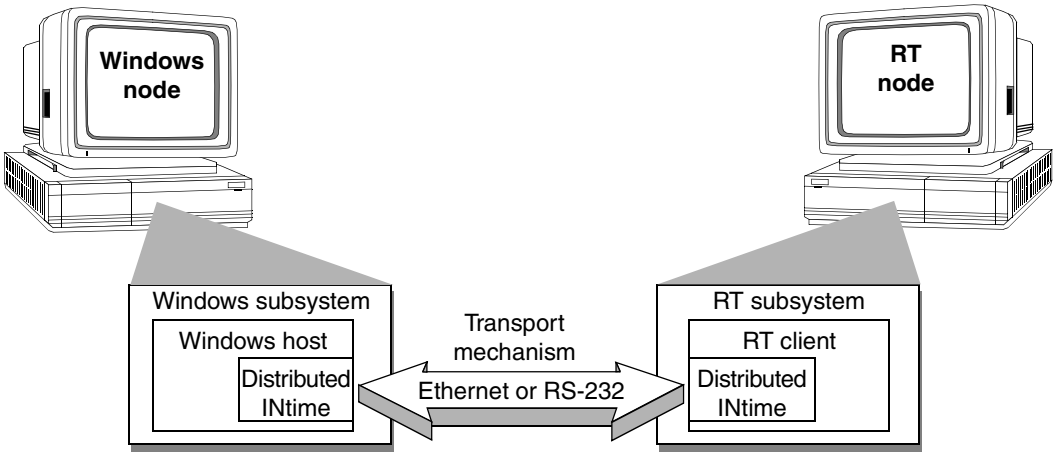
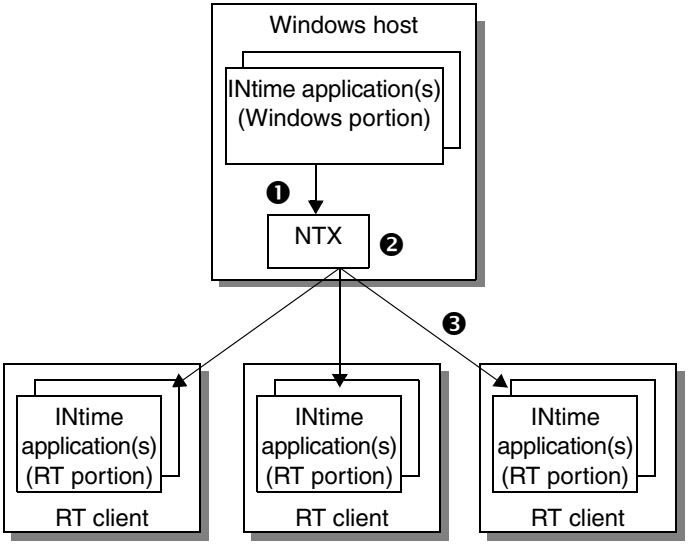


Figure 2-3. How INtime runs between computers

Transport mechanisms

With INtime software, NTX communicates between Windows and RT portions of INtime applications whether they reside on a single PC, or on separate computers accessed via serial or Ethernet cable:



❶ The Windows portion of INtime applications, located on a Windows host, makes NTX calls that communicate to RT clients.

❷ NTX determines RT client locations, detects the connection method, and determines how to communicate between Windows and RT threads.

❸ NTX uses the appropriate transport method to communicate with the RT portion of the INtime applications, located on RT clients.

Figure 2-4. How NTX communicates with RT nodes

Transport methods available to NTX include:

Transport mechanism	Transport driver	Description
OSEM	RTIF.SYS	Used when the Windows host and RT client co-exist on a single PC. For details, see *About the OSEM* and *How the RT interface driver works* later in this chapter.
Ethernet	UDP/IP	Used for Windows hosts and RT nodes connected via a local area network (LAN) cable. For details, see *About remote NTX* later in this chapter.
RS-232	Serial I/O	Used for Windows hosts and RT nodes connected by serial cable. For details, see *About remote NTX* later in this chapter.

About the OSEM

The OSEM manages the simultaneous operation of Windows and the RT kernel on the same CPU. It encapsulates all of Windows as an RT thread, and then transparently switches execution to the appropriate kernel, based on interrupt activity and thread scheduling. Once encapsulated, Windows (with all its processes and threads) execute as a single, low priority, RT thread in the context of the RT root process.

The OSEM provides:

- **Isolated processes**: Uses standard Intel architecture support for hardware multi-tasking to maintain proper address space isolation and protection between Windows processes and RT processes. This approach also ensures RT responsiveness, regardless of Windows activity.
- **Transparent thread creation and switching**: Transparently creates a hardware task for the RT kernel, and manages the switching and execution of both the standard Windows and INtime system hardware tasks.

  In a standard Windows configuration, the bulk of the OS runs in the confines of a single hardware task. Additional hardware tasks are defined only to handle catastrophic software-induced failures, such as stack faults and double faults, where a safe and known environment is required from which to handle the failure. INtime software's task switching approach guarantees the integrity of both Windows and the RT kernel, and enables the successful operation of RT threads even in the event of a total Windows failure (a blue screen crash).
- **Additional address isolation via 32-bit segmentation**: Provides additional address isolation and protection between complex RT processes, and between RT processes and Windows code. The RT kernel accomplishes this by using multiple sets of 32-bit segments, separate from those used by Windows.
- **Easy-to-use interface**: Provides a clean, well defined interface, which minimizes interaction with Windows to a few key areas. The result is improved product reliability and simplified compatibility between Windows releases.

With INtime applications running on a single PC, the INtime runtime environment encapsulates all Windows processes and threads into a single RT thread of lowest priority as shown in the next figure. As a result, RT threads always preempt running Windows threads, guaranteeing hard determinism for all RT activities within the system.

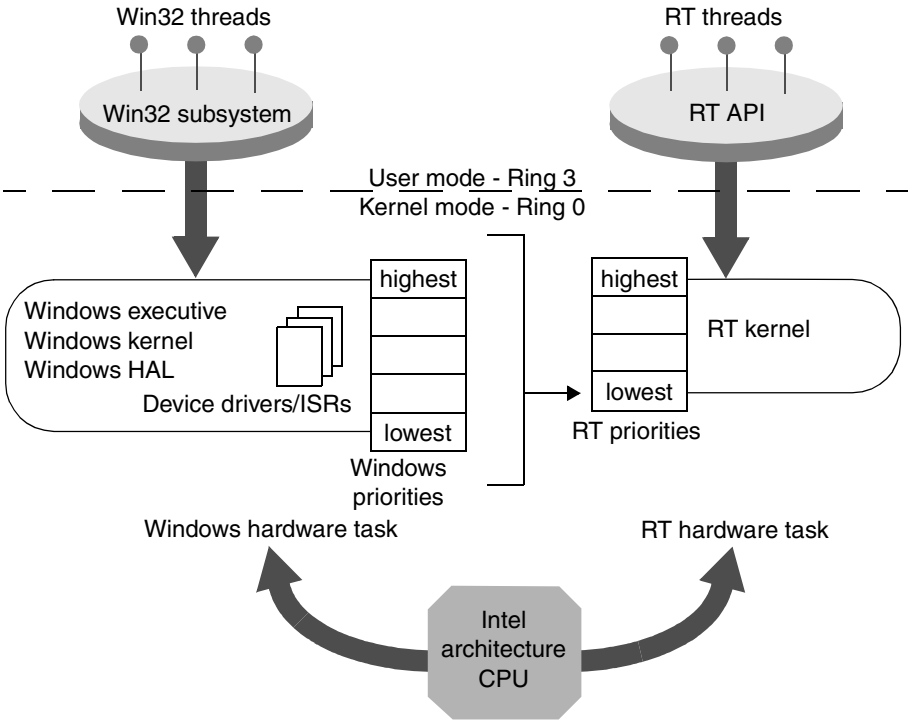


Figure 2-5. Encapsulating Windows processes and threads into an RT thread

When an interrupt occurs, the INtime runtime environment responds in one of these ways:

Interrupt type	Windows in control	RT in control	Shared control
Windows	Windows maintains control.	RT maintains control.	RT determines whether to maintain or relinquish control.
RT	RT takes control, pre-empting Windows activity.	RT maintains control.	RT maintains control.

How the RT interface driver works

RTIF.SYS is a Windows device driver that provides centralized support for the OS encapsulation mechanism (OSEM). The RT Interface Driver facilitates communications between RT kernel threads and Windows threads.

The RTIF driver begins execution as a Windows system service, early in the Windows boot process. During initialization, it allocates physically contiguous memory for the RT kernel's memory pool.

The RTIF driver cooperates with the RT Application Loader to load and start the RT kernel in its own environment. The driver queries the registry for various kernel parameters and passes them to the RT kernel at the kernel's initialization time. Parameters include:

- The number of Windows threads that can simultaneously make NTX library calls.
- The low-level tick duration used by the RT kernel.
- The Windows time base currently in effect (in microseconds).

If the RT kernel is running, the RTIF driver:

- Routes the clock interrupt (IRQ 0), based on who needs the next clock tick, to either the RT kernel or to the Windows clock interrupt entry point for processing.

  When neither environment needs the tick, the driver sends an EOI to the PIC for this level and returns control to the interrupted Windows thread.
- Immediately routes all other real-time interrupts to the RT kernel for processing.
- Relays NTX library requests to the RT kernel and blocks the calling Windows thread until the RT kernel responds to the request and/or until resources are available to complete the request.

  Otherwise, the RTIF driver terminates NTX library requests. When the RT kernel announces its termination, the RTIF driver terminates all pending requests.
- Manages the Windows portion of controlled shutdown during a Windows blue screen crash: the handler notifies the RT kernel to handle the RT portion of the controlled shutdown. If the kernel is not running, control is returned to Windows.

In summary, the RTIF.SYS device driver contains the Windows portion of the OSEM. It also acts as the NTX transport driver for a co-resident, or local, RT kernel. RTIF.SYS allocates physical memory for the RT kernel and locks that memory in place so it will not be used or paged to disk by the Windows kernel. A Windows service loads the RT kernel into the allocated memory and issues a "start kernel" command to RTIF.SYS. In response to the start command, the driver establishes a separate hardware task for the RT kernel and hands off control to the kernel's initialization code. After initializing its environment, the RT kernel creates a low-priority thread (priority level 254) which returns to Windows and becomes the Windows thread.

About remote NTX

The INtime RT kernel may be installed and run on a standalone PC platform, and connected to the Windows workstation. Such an installation is called an "RT Node". Communication between a Windows application and an RT application running on the RT node uses the same NTX interface as with the local kernel, with Ethernet or a serial line as the communications medium. All the calls are available and work identically, except for shared memory.

About the modified Windows HAL

INtime software uses the Windows HAL, but modifies it in place to perform the following actions:

- Traps attempts to modify the system clock rate so that the RT kernel can control the system time base.
- Traps attempts to assign interrupt handlers to interrupts reserved for RT kernel use.
- Ensures that interrupts reserved for RT kernel use are never masked by Windows software.

INtime software is compatible with the following HAL files:

- "Standard PC"
- "Advanced Configuration and Power Interface (ACPI) PC"

On INtime-compatible PCs, other HAL files may be replaced with one of these. For details, see *Chapter 6, Installation*.

About thread scheduling

The RT kernel switches between threads and makes sure the processor always executes the appropriate thread. The kernel's scheduling policy is that the highest priority thread that is ready to run is/becomes the running thread. The kernel maintains an execution state and a priority for each thread and enforces its scheduling policy on every interrupt or system call.

Priority-based scheduling

A priority is an integer value from 0 through 255, with 0 being the highest priority.

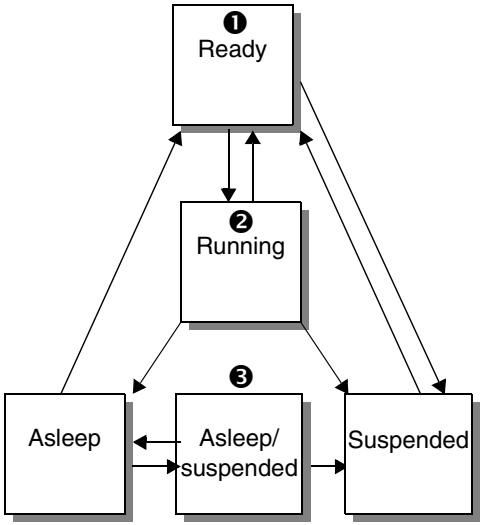
Range	Usage
0–127	Used by the OS for servicing external interrupts. Creating a thread that handles internal events here masks numerically higher interrupt levels.
128–130	Used for threads that communicate with interrupt threads.
131–253	Used for threads that handle internal events, such as message passing. You can usually start using round-robin scheduling at about priority 141.

Interrupt threads mask lower-priority (numerically higher) interrupt levels. When you assign interrupt levels, give a higher-priority (numerically lower) level to interrupts that can't wait, such as serial input, and a lower priority (numerically higher) to interrupts that can wait, such as cached input.

Execution state

The execution state for each thread is, at any given time, either running, ready, asleep, suspended, or asleep-suspended. The RT kernel enforces the scheduling policy in which the highest priority ready thread is always the running thread.

Threads run when they have the highest (numerically lowest) priority of all ready threads in the system and are ready to run. Threads can change execution state, as shown in the next figure.



❶ Threads are created in the ready state.

❷ The running thread, the ready thread with the highest priority, does one of these:

- Runs until it removes itself from the ready state by making a blocking system call.
- Runs until its time slice expires (when running with a priority lower—numerically higher—or equal to the configured round robin threshold priority with other threads at the same priority.
- Runs until preempted by a higher priority thread which has become ready due to the arrival of an interrupt, or through the receipt of a message/unit at an exchange at which the higher priority thread was blocked.

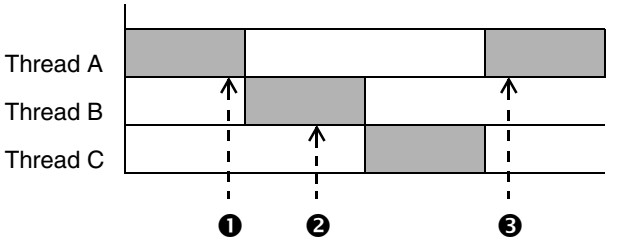
❸ A thread in any state except ready cannot run, even if it has the highest priority.

Figure 2-6. Execution state transitions for threads

A thread can put itself to sleep or suspend itself by using system calls for that purpose. The RT kernel might indirectly put a thread to sleep if the thread makes a "blocking" call by, for example, waiting at a mailbox until a message arrives. The kernel puts the thread in the ready state when the message arrives.

Round-robin scheduling

INtime software also provides round-robin scheduling, where equal-priority threads take turns running. Each thread gets a time slice. If a thread is still running when its time slice expires, that thread moves to the end of a circular queue for that priority level where it waits until all threads ahead of it use up their time slices, as shown in the next figure. You adjust the length of the time slice and set the priority level where round-robin scheduling occurs.



Threads A, B, and C are of equal priority below the round-robin priority threshold.

❶ Thread A, the running thread, stops running when its time slice runs out. Thread A's state is saved and it moves to the end of the queue.

❷ Thread B, a ready thread, then becomes the running thread.

❸ Thread A runs again when all threads in the queue either finish running or are preempted when their time slice expires.

Figure 2-7. Round-robin scheduling

Higher-priority threads still preempt any running thread in the round-robin queue, regardless of the amount of time left in its time slice.

Round-robin scheduling cannot guarantee a predictable worst-case response to events because the number of threads in the queue varies.

Handling interrupts

System hardware invokes an interrupt handler in response to an asynchronous interrupt from an external source, based on its entry number in the IDT (Interrupt Descriptor Table). The handler takes control immediately and saves the register contents of the running thread so it can be restarted later. There are two ways you can service an interrupt:

- Using a handler alone
- Using a handler/thread combination

Interrupt handler alone

An interrupt handler alone can process only interrupts that require very little processing and time. Handlers without threads can:

- Accumulate data from the device in a buffer. The data must have an associated thread to process the data.
- A handler begins running with all interrupts disabled. It must execute quickly and then exit to minimize its effect on system interrupt latency.
- Find the interrupt level currently serviced. This is useful if one handler services several interrupt levels.
- Send an EOI (End of Interrupt) signal to the hardware.

By itself, an interrupt handler can only do very simple processing, such as sending an output instruction to a hardware port to reset the interrupt source. Handlers can use only a few system calls. For a list and description of system calls, see *Appendix A*, "*INtime software system calls*".

During the time the interrupt handler executes, all interrupts are disabled. Since even very high level interrupts are disabled, it is essential that the handler execute quickly and exit.

When the handler finishes servicing the interrupt, it sends an EOI to the PIC (Programmable Interrupt Controller) via an INtime software system call, restores the register contents of the interrupted thread, and then returns to the interrupted thread.

Interrupt handler/thread combination

An interrupt handler/thread combination provides more flexibility. Although the handler may perform some processing, it typically signals the corresponding interrupt thread to do most or all interrupt processing. In general, use an interrupt handler/thread combination if the processing requires more than 50 microseconds or requires system calls that interrupt handlers cannot use.

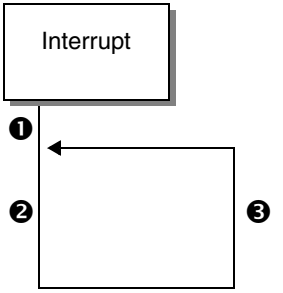
When an associated interrupt thread exists, the handler can put accumulated information into a memory address, if the interrupt thread has set one up. The interrupt thread can access data in the memory address and perform the required processing.

Interrupt threads have access to the same resources and use the same system calls as ordinary threads. The RT kernel assigns an interrupt thread's priority, which is based on the interrupt level associated with the handler. Ordinary threads have a priority assigned by the process.

In addition to the usual thread activities, an interrupt thread can also:

- Cancel the assignment of an interrupt handler to an interrupt level.
- Wait for an interrupt to occur.
- Enable and disable interrupts.

This shows how an interrupt thread enters an event loop where it waits to service an interrupt:



❶ Upon creation, the interrupt thread uses an RT system call to set up an RT interrupt and associate itself with this interrupt. Normally, it then waits for a signal that indicates an interrupt occured.

❷ When signaled, the interrupt thread executes the required operations.

❸ The interrupt thread releases control by waiting for the next signal from the interrupt handler, which restarts the cycle shown in this figure.

Figure 2-8. Thread execution model

Managing time

INtime software enables threads to:

- Create single-shot and repetitive alarm events for a specified time interval, and specify which alarm event thread to activate.
- Start and stop scheduling by the RT kernel.

Alarm events

Software alarm events are virtual timers that invoke alarm event threads that you write.

3 About INtime software's RT kernel

This chapter describes objects provided by the RT kernel.

What does the RT kernel provide?

The RT kernel provides:

Object management	Includes creating, deleting, and manipulating object types defined by the kernel. Memory for high-level kernel objects is automaticallly taken from your processor's memory pool. You must provide memory for low-level kernel objects and may allocate memory beyond the kernel's needs to store application specific state information associated with the low-level object.
Time management	Includes an RT clock, alarms that simulate timer interrupts, and the ability to put threads to sleep.
Thread management	Includes scheduling locks which protect the currently running thread from being preempted.
Memory management	Implements memory pools from which it allocates memory in response to application requests.

RT kernel objects

Objects, data structures that occupy memory, are building blocks that application programs manipulate. Each object type has a specific set of attributes or characteristics. Once you learn the attributes of, for example, a mailbox, you know how to use all mailboxes.

Object-based programming, which concentrates on objects and operations performed on them, is compatible with modular programming. Typically a single thread performs only a few related functions on a few objects.

The RT kernel provides basic objects and maintains the data structures that define these objects and their related system calls. When you create an object, the RT kernel returns a handle that identifies the object:

- **High-level objects** consume memory, but also a slot in the system GDT (Global Descriptor Table). Therefore, the maximum number of high-level objects allowed in the system at any one time is approximately 7600 (8192 slots in a GDT minus slots used by the operating system).
- **Low-level objects** consume only memory. Therefore, only the amount of system memory controls how many low-level objects can be present at a given time.

The RT kernel provides these objects. Each object is discussed on the indicated page:

Object	Description	Page
Threads	Do the work of the system and respond to interrupts and events.	28
Processes	Environments where threads do their work.	28
Exchange objects	Used by threads to pass information.	32
Mailboxes	Used by threads to pass objects and data. INtime software includes both object and data mailboxes.	33
Semaphores	Used by threads to synchronize.	34
Regions	Used by threads to provide mutual exclusion.	35
Ports	Used by threads to synchronize operations, pass messages, and access INtime services.	36
Dynamic memory	Addressable blocks of memory that threads can use for any purpose.	30

For detailed information about RT kernel objects, how they operate, and system calls associated with each object, see INtime Help.

Threads

Threads, or threads of execution, are the active, code-executing objects in a system.

Threads typically respond to external interrupts or internal events. External interrupts include events such as a keystroke, a system clock tick, or any other hardware-based event. Internal events include events such as the arrival of a message at a mailbox. Threads have both a priority and an execution state, whether the thread is running or not.

There are system calls to create and delete threads, view and manipulate a thread's priority, control thread readiness, and obtain thread handles. For a list and description of these system calls, see *Appendix A*, "*INtime software system calls*".

Processes

A process is an RT kernel object that contains threads and all their needed resources. Processes make up your INtime applications. The RT kernel processes have these characteristics:

- Cannot make system calls ; they are passive.
- May include one or more threads.

- Isolate resources for their threads, particularly for dynamically allocated memory. Two threads of one process compete for the memory associated with their process. Threads in different processes typically do not.
- Provide error boundaries. Errors within one process do not corrupt other processes or the OS because they reside in separate virtual address spaces.
- When you delete processes, the objects associated with them also are deleted.

Each INtime application's executable loads as a separate, loadable process. The processes in a system form a process tree. Each application process obtains resources from the root:

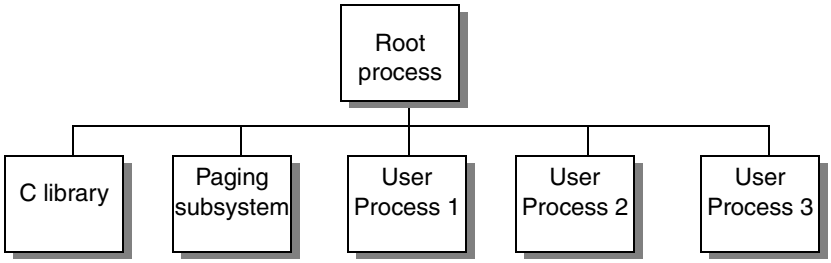


Figure 3-1. Processes in a process tree

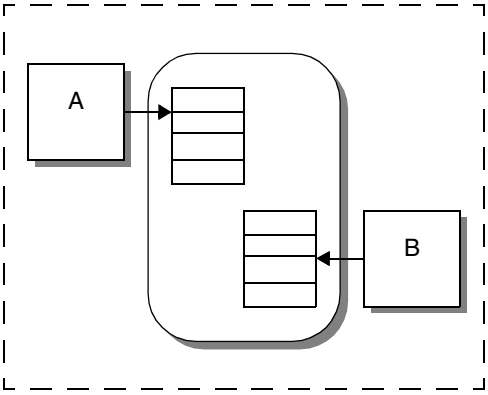
The RT Application Loader creates RT processes when an INtime application loads. There are system calls you can use to delete RT processes from within an application.

Virtual memory

Each process has an associated VSEG whose size is the amount of Virtual Memory available to the process. The VSEG size must be large enough to contain all the memory dynamically allocated by the threads within the process.

Memory pools

Each process has an associated memory pool, an amount of memory with a specified minimum and maximum, allocated to the process. Minimum memory is always contiguous. Usually, all memory needed for threads to create objects in the process comes from the process's memory pool, as shown in the next figure.



Threads A and B obtain memory from the process's memory pool.

Figure 3-2. Threads using their process's memory pool

If not enough contiguous memory exists (up to the maximum size of the process's memory pool), the RT kernel tries to borrow from the root process.

You can also statically allocate memory to processes, but you cannot free memory allocated in this manner for other processes. The system's total memory requirement is always the sum of the memory requirements of each process. Static memory allocation uses more memory than dynamic allocation, but may be safer.

Dynamic memory

Dynamic memory supports many uses, including communicating and storing data. The memory area is usually allocated from the memory pool of the thread's process, as shown in *Figure 3-2."Threads using their process's memory pool"*. If there is not enough memory available (up to the maximum size of the process's memory pool), the kernel tries to borrow from the root process.

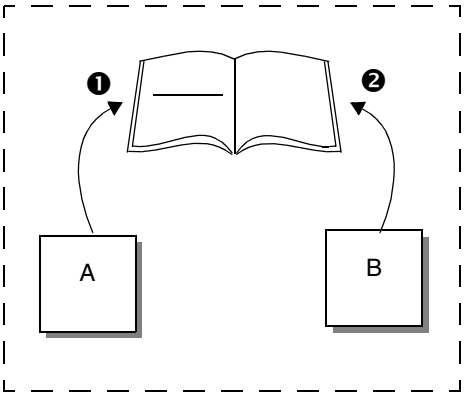
INtime software includes system calls that allocate and free memory and create handles for allocated memory to share with other processes. For an overview of these calls, see *Appendix A, INtime software system calls*. For detailed information, including syntax and parameter values, see Help.

Object directories

Each process has an associated object directory. When a thread creates an object, the RT kernel creates a handle for it. A thread can catalog a high-level object, with its handle and a corresponding name, in the object directory of its own process or any other process it knows about. Typically, you catalog objects in the root directory so that threads in other processes can access them.

Threads in the same process and threads that know the name can use the object directory to look up and access objects, or they can use global variables within their address space to identify and access system objects contained within their process.

You cannot catalog the handle for a low-level object in an object directory.



❶ Thread A catalogs an object such as a data mailbox in the root process's object directory.

❷ Thread B looks up the object in the object directory to use it.

Now thread A can send data to the mailbox and thread B can receive it.

Figure 3-3. Threads using the root process's object directory

Exchange objects

Validation levels

INtime software provides two levels of system calls for exchange objects:

Table 3-1. Exchange objects: validation levels

Level	Description	Exchange objects
High (validating)	Provides lower performance and higher protection and validation features. Memory is allocated automatically from the process's pool. High-level objects: • Validate parameters. • Are protected against unexpected deletion.	• Object mailboxes • Data mailboxes • Counting semaphores • Regions (for mutual exclusion with priority inversion protection)
Low (non-validating)	Provide higher performance and lower protection and validation features. Low-level objects provide functionality beyond that of high-level objects. You must allocate memory for low-level objects and may allocate memory beyond low-level object needs. You can use this additional memory to store application-specific state information associated with the object. Low-level objects: • Do not validate parameters. If you need parameter validation, use high-level system calls instead. • Are not protected against unexpected deletion. **Note**: System calls that manipulate low-level objects assume that all memory reference pointers received are valid.	• Data mailboxes • Single-unit semaphores • Region semaphores (with priority inversion protection) • Software alarm events (virtual timers) that invoke alarm event threads that you write.

Write, test, and debug your application using high-level calls with their protection and validation features. When the application runs as you desire, increase performance by substituting low-level system calls where appropriate.

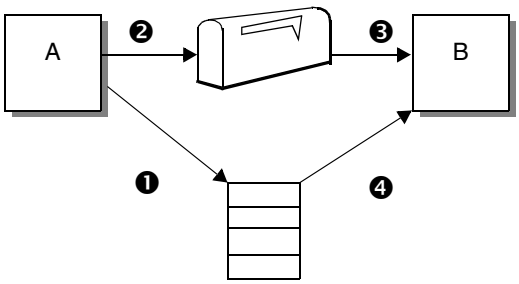
For more information about validation levels, see RT calls in *Appendix A, INtime software system calls*.

Mailboxes

Mailboxes provide communication between threads in the same process or in different processes. They can send information and, since a thread may have to wait for information before executing, they can synchronize thread execution. There are two mailbox types:

- **Object mailboxes**: Send and receive object handles. Available only as high level objects.
- **Data mailboxes**: Send and receive data. Available as both high- and low-level objects. High-level data mailboxes have a maximum limit of 128 bytes.

The next figure shows how threads use an object mailbox to send a handle for a memory address.



❶ Thread A allocates a block of memory and creates a shared-memory handle for it. Data is placed in this shared memory object.

❷ Thread A sends the shared memory handle to a mailbox.

❸ Thread B waits to receive the shared memory handle at the mailbox. You can specify whether or not thread B should wait if the handle isn't in the mailbox.

❹ Thread B obtains the handle and accesses the data in the memory object by mapping the memory associated with the memory object into its memory address space.

Figure 3-4. Threads using an object mailbox

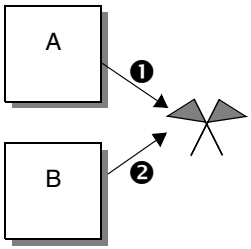
Mailboxes have thread queues, where threads wait for messages, and message queues, where messages wait threads to receive them. The thread queue may be FIFO- or priority-based; the message queue is always FIFO-based.

You use the same system calls to create and delete object and data mailboxes. However, you use different calls to send and receive messages or data.

Semaphores

A semaphore is a counter that takes positive integer values. Threads use semaphores for synchronization by sending units to and receiving units from the semaphores. When a thread sends *n* units to a semaphore, the value of the counter increases by *n*; when a thread receives *n* units from a semaphore, the value of the counter decreases by *n*.

The next figure shows a typical example of a binary (one-unit) semaphore used for synchronization.



To ensure that thread A can do its work before thread B starts running, thread A creates a semaphore that contains one unit. To enable synchronization, threads A and B should request and obtain the unit before running.

❶ Thread A begins to run and obtains the semaphore unit, leaving the semaphore empty. While the semaphore has no units, thread B cannot run.

❷ When thread A completes, it returns the unit to the semaphore. Thread B can now obtain the unit and start running.

Figure 3-5. Threads using a semaphore for synchronization

Semaphores:

- Enable synchronization; they don't enforce it. If threads do not request and obtain units from the semaphore before running, synchronization does not occur. Each thread must return the unit to the semaphore when it is no longer needed. Otherwise, threads can be permanently prevented from running.
- Provide mutual exclusion from data or a resource as follows:

  1. Thread A requests one unit from a binary semaphore, and uses the resource when it receives the unit.
  2. Thread B requests one unit from the semaphore before using the resource. Thread B must wait at the semaphore until thread A returns the unit.

- Enable mutual exclusion; they do not enforce it.
- Have a queue where threads wait for units. The queue may be FIFO- or priority-based. There are system calls to create and delete semaphores, and to send and receive units.

Regions

A region is a single-unit semaphore with special suspension, deletion, and priority-adjustment features. Regions provide mutual exclusion for resources or data; only one thread may control a region at a time; only the thread in control of the region can access the resource or data protected by a region. Once a thread gains control of a region, the thread cannot be suspended or deleted until it gives up control of the region. When the running thread no longer needs access, it exits the region, which enables a waiting thread to obtain control of the region and thus access the resource or data protected by that region.

Regions have a thread queue where threads wait for access to the region. The queue may be FIFO- or priority-based.

Regions also have a priority-inversion avoidance mechanism when the region's thread queue is priority based.

Then, if a higher-priority thread tries to enter a busy region, the priority of the thread in the region is raised temporarily so that it equals the waiting thread's priority. This helps prevent priority-inversion, as shown in this example:

1. Thread A is the running thread. It is a low-priority thread with control of a region, accessing some data. The region has a priority queue. The only other thread that uses the data is thread C, a high-priority thread that is not ready to run.
2. Thread B, a medium-priority thread, becomes ready to run and preempts A.
3. Thread C becomes ready to run and preempts B. It runs until it tries to gain control of the region. Thread A's priority is raised to equal thread C's priority until thread A releases the region; then its priority returns to its initial level.
4. When thread A releases the region, thread C receives control of the region and uses the data. When thread C completes, thread B runs.

Without the priority inversion avoidance mechanism, thread B would have preempted A while A had control of the region; C would have preempted B, but would have been unable to use the data because A had control of the region.

Regions require careful programming to avoid deadlock, where threads need simultaneous access to resources protected by nested regions, and one thread has control of one region while the other thread has control of another. To avoid deadlock, all threads must access nested regions in the same, arbitrary order, and release them in the same reverse order.

Ports

A port is the object which allows access to the features provided by an INtime service. A process that uses a port object can send messages through the port to the INtime service, or can receive messages through the port from the service. Other operations possible on ports include:

- Attach a heap object to the port for use by the service to store received messages.
- Link ports to a sink port, allowing a single thread to service multiple ports.

Services

An INtime service is an INtime real-time application (RTA) which provides access to one or more interfaces. Each interface is associated with a service descriptor. The interface generates events which are handled by the service. A process which uses a service creates a port for access to that service. A service may support more than one port and more than one user process may use a given port. A user process communicates with the service by sending and receiving messages via the port.

Heaps

A heap is an INtime memory object that manages the chunk of dynamic memory allocated to it. A heap can be used by multiple processes that need to share large amounts of information. For instance, a heap can be associated with a port. Data placed in memory obtained from the heap by threads in one process (the thread using a port to communicate with an INtime service) can be manipulated by threads in another process (thread within the service accessing data passed through the port to the service).

4 About RT programming

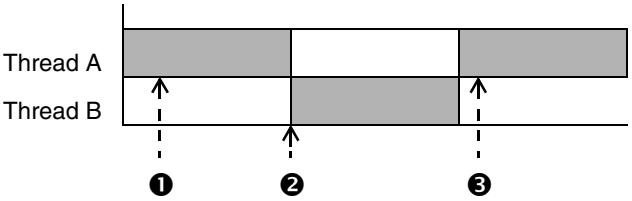
This chapter describes mechanisms unique to RT programming:

For information about developing an RT application using INtime software, see *Chapter 10, INtime application development*.

Mechanism	Description	Page
Multi-threading	Switches between threads of execution.	38
Preemptive, priority-based scheduling	Determines which thread needs to run immediately and which can wait.	39
Interrupt processing	Responds to external interrupts that occur during system operation.	40
Determinism	Enables threads to execute in a predictable fashion, regardless of the arrival of both internal and external events.	41
Multi-programming	Allows more than one application to run at a time.	42
Inter-thread coordination and communication	Enables asynchronous threads, which run in a random order, to coordinate and communicate with one another	43
Messages	Enables threads to exchange data, messages, or object handles.	44
Synchronization	Enables threads to signal the second thread when a task is completed.	45
Mutual exclusion	Prevents threads from accessing data currently in use until released.	45
Memory pools and memory sharing	Allocates memory to RT applications on request and manages multiple memory requests.	46
System calls	Programmatic interfaces you use to manipulate objects or control the computer's actions.	48
Real time shared libraries	Libraries you build that can be shared by one or more real-time applications.	48
Exception handling	Causes and proper handling of system exceptions.	49

Multi-threading

Multithreading means the computer stops running one thread and starts running another, as shown in the next figure. INtime software manages thread switching, saving the old thread's context on the old thread's stack and loads the new thread's context before starting execution. An INtime software thread is a thread of execution, similar to a Windows thread.



❶ The processor executes thread A.

❷ An event happens and a thread switch occurs. The processor then executes thread B.

❸ When thread B finishes, thread A becomes the running thread again.

Figure 4-1. Thread switching in a multithreading environment

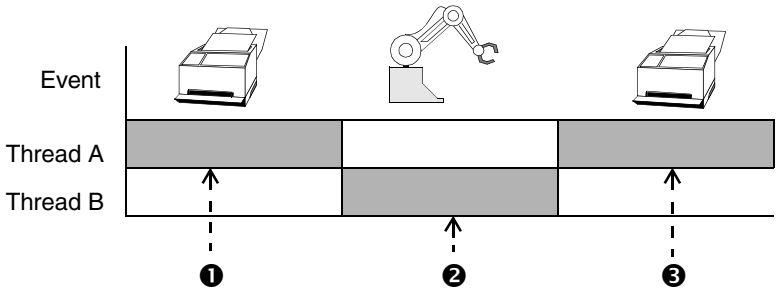
Multithreading and modular programming go hand-in-hand. You start by breaking down a large, difficult application problem into successively smaller and simpler problems, grouping similar problems where you can. Finally, you solve the small problems in separate program modules. In the INtime software multithreading environment, each module is a thread.

Multithreading simplifies building an application. When you need a new function, you just add a new thread.

When you combine multithreading with preemptive, priority-based scheduling, your application can switch as appropriate: from relatively unimportant threads, to important threads, to critical threads, and back again.

Preemptive, priority-based scheduling

In a preemptive, priority-based system, some threads are more critical than others. Critical threads run first and can preempt less critical threads, as shown in this figure:



❶ Thread A, a low-priority thread, prints data accumulated from the robotic arm in report form.

❷ Thread B, a high-priority thread, controls the robotic arm. If the arm needs to move while thread A runs, thread B preempts the print thread, then starts and moves the arm.

❸ After thread B repositions the arm, thread A finishes printing.

Figure 4-2. Multithreading and preemptive, priority-based scheduling

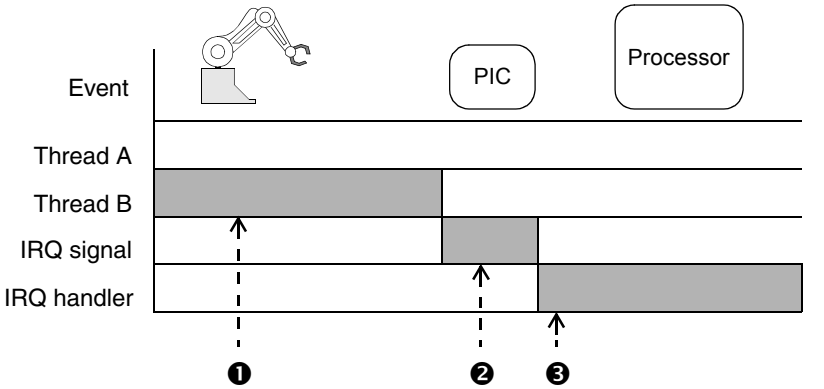
Multithreading allows an application to respond to internal events and external interrupts, such as clock ticks from the system clock or receiver ready from a serial device, based on how critical they are. You determine the priority of threads in your application; INtime software provides the thread scheduling algorithms.

When you add interrupt processing to multithreading and preemptive, priority-based scheduling, your application can respond to interrupts as they occur. Your application becomes event-driven.

Interrupt processing

Interrupts are signals from devices such as a malfunctioning robot or interactive terminal. You connect interrupt sources to the processor through the PC's two PICs (Programmable Interrupt Controllers).

With interrupt processing, your application can handle interrupts occurring at random times (*asynchronously*) and can handle multiple interrupts without losing track of the running thread, or those threads waiting to run. Interrupts can occur while the processor is executing either an unrelated thread or a related thread, as shown in the next figure.



❶ Thread B, the running thread, repositions the robotic arm.

❷ The robotic arm malfunctions and sends an interrupt signal through the PIC.

❸ As soon as it receives the signal, the microprocessor stops the running thread and starts an interrupt handler. The interrupt handler runs in the context of thread B. No new thread is loaded; thread B's state does not need to be saved. It remains loaded in RAM until the scheduler runs it again. Thread A, the print thread, is still waiting to run.

Figure 4-3. Interrupt handler interrupting a thread

Typically, numerous sources of interrupts exist in an application. Some of them, like the malfunctioning robotic arm, are critical; some of them are not. You assign interrupt levels (which map directly to priorities) to the interrupt sources by the order in which you connect your external sources to the PIC. INtime software handles more critical interrupts first, and keeps track of which interrupts occurred, the order in which they occurred, and which ones have not been handled.

Interrupt handlers can perform very limited operations, so you typically write an interrupt handler to signal an interrupt thread. The interrupt thread's priority can be automatically assigned, based on the interrupt level of the external source.

Multithreading and interrupt processing simplify expanding an application. Because of the one-to-one relationship between interrupts and threads, you add a new thread when you need to respond to a new interrupt. Interrupt processing is also more efficient, since your system spends all of its time running threads, not polling for interrupts.

Determinism

INtime software provides deterministic response by establishing a predictable, worst-case response time to a high-priority interrupt. Deterministic response time includes these components:

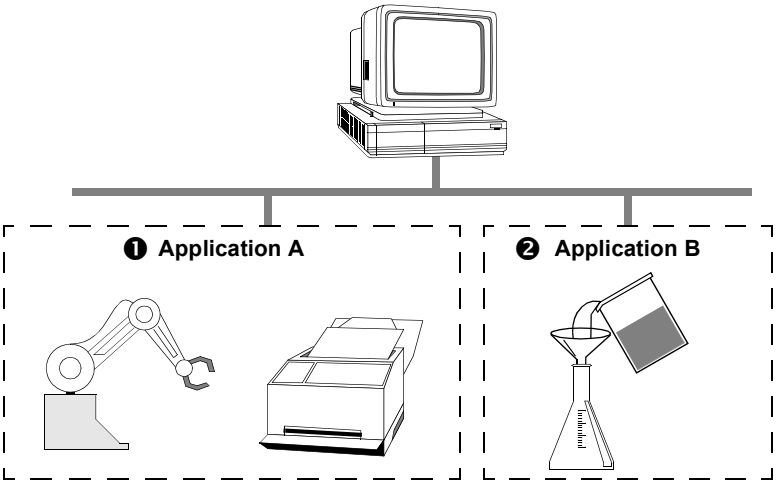
- **Interrupt response time**: The time that elapses between a physical interrupt and the start of interrupt handler execution. A predictable worst-case response time to interrupt processing ensures that incoming data is handled before it becomes invalid.
- **Thread switch time**: The time that elapses between exiting one thread and starting another. To exit a thread, the RT kernel must save data registers, stack and execution pointers (the thread state) of one thread. Minimized thread switch time also provides a predictable response time to a high-priority thread.

Since the typical response to an interrupt includes invoking a handler and then performing a thread switch to an interrupt thread, the deterministic response time includes both the interrupt response and thread switch times.

RT response does not mean instantaneous execution. A high-priority thread that is very long and performs many calculations uses as much processor time to execute on an RT system as on any other system. The length of time instructions take to execute is a function of processor speed.

Multi-programming

INtime software supports multiprogramming—running several unrelated applications on a single system at the same time.

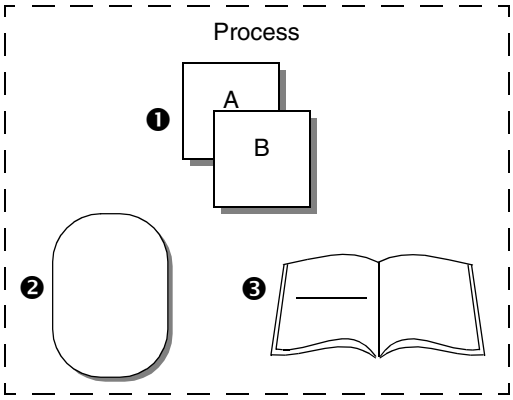


❶ Application A contains all the threads that relate to a robotic arm, including the print thread. It may also contain threads that control other devices on the factory floor.

❷ Application B contains all the threads that relate to an application that controls a chemical mixing system in one part of the factory.

Figure 4-4. Multiprogramming

To take full advantage of multiprogramming, you provide each application with a separate environment: separate memory and other resources. INtime software provides this kind of isolation in a process. Typically, a process includes a group of related threads and the resources they need, as shown in the next figure.



❶ A group of related threads.

❷ The memory the threads need.

❸ An object directory where you can catalog thread resources.

Figure 4-5. Resources in a process

You decide what processes to include in your system. INtime software coordinates the use of resources within and between processes so independently-developed applications do not cause problems for each other.

Multiprogramming simplifies adding new applications; you can modify your system by adding new processes (or removing old ones) without affecting other processes.

Inter-thread coordination and communication

INtime software exchange objects are mailboxes, semaphores, regions, and message ports. They enable asynchronous threads, which run in a random order, to coordinate and communicate with one another by:

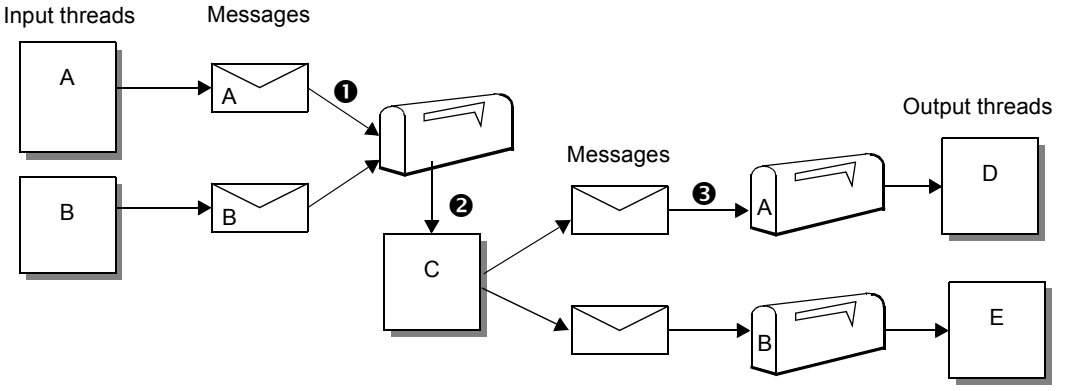
- Passing messages
- Synchronizing with each other
- Mutually excluding each other from resources

Messages

Threads may need to exchange data, messages, or object handles.

For example, a thread accumulates input from a terminal until it receives a carriage return. The thread then uses an exchange object to send the entire line of input as data to another thread that decodes the input.

This figure summarizes how you can solve a problem that requires routing several input types into several output types using a mailbox object. One mailbox and one manager thread can handle messages from multiple input and output threads.



❶ System calls move data from input threads A and B to a waiting mailbox.

❷ Thread C, the manager thread, waits at the mailbox and determines which messages go to which output threads. If another message arrives during processing, the message waits in the mailbox queue until the manager thread can handle it.

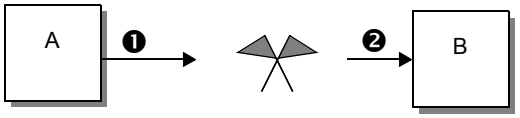
❸ The individual output threads receive data at their mailboxes and execute it.

Figure 4-6. Object-based solution for message passing

Synchronization

When one thread needs to run before another thread, it can use an exchange object to signal the second thread when it has completed. For example, the thread that creates the transaction summary in an automated teller application shouldn't run until after the threads that handle withdrawals and deposits run. The transaction summary thread must synchronize with the other threads.

INtime software provides several objects for synchronization that accommodate a wide variety of situations. The next figure illustrates using a semaphore to send a signal to another thread.



Thread A, the running thread, preprocesses some data. Thread B needs to use the data after thread A finishes.

❶ When thread A finishes, it sends a signal (not data) to the semaphore.

❷ When thread B receives the signal, it processes the data.

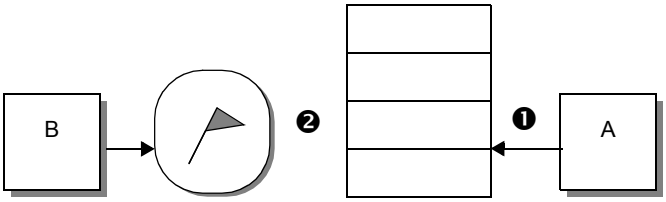
Figure 4-7. Threads that use a semaphore for synchronization

Mutual exclusion

INtime software includes regions that you can use to protect data from being accessed by multiple threads at the same time. This is called mutual exclusion.

When an INtime application runs, multiple threads can concurrently access the same data. This is useful in a multithreading system, such as a transaction processing system where a large number of operators concurrently manipulate a common database.

If a distinct thread drives each client, an efficient transaction system requires that threads share access to database data. When threads run concurrently, this situation ocasionally arises:



❶ Thread A, the running thread, reads data from the database and performs computations based on the data.

❷ Thread B tries to preempt thread A and update the data while thread A works on it. Mutual exclusion, provided by a region, prevents two threads from accessing the same data concurrently.

Figure 4-8. Multithreading and mutual exclusion

Unless thread B is prevented from modifying the data until after thread A has finished, thread A may unknowingly use some old data and some new data, resulting in an invalid computation. It should, however, read and compute the new data after thread B updates it.

Memory pools and memory sharing

Memory pools are the basis of INtime software's memory management. Two types of memory pools exist:

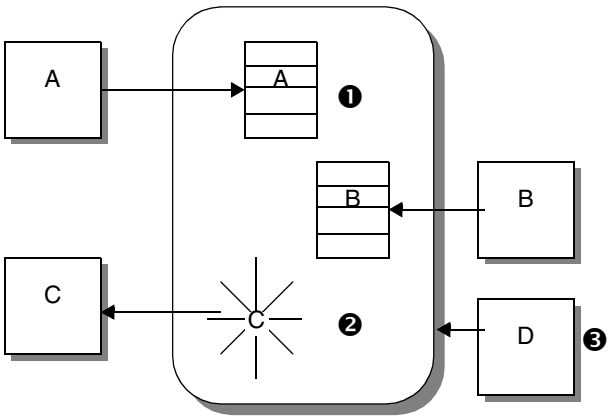
- **Initial memory pool**: All the memory available to the RT kernel (that is, free space memory). Managed by the RT kernel, the initial memory pool belongs to the root process and is allocated to INtime applications on request. Any amount of memory you allocate to the RT kernel is not available to Windows for that session. There is a practical maximum for the amount of memory allocated to INtime software. This value depends on the number and size of INtime applications that run on the RT kernel.
- **Process memory pools**: A portion of the initial memory pool assigned to an INtime application process when that process is created. Each process memory pool has a minimum and a maximum size. Once the minimum memory is allocated to a process, that memory is not available to other processes. When you delete a process, its memory returns to the initial memory pool.

  As threads in a process create and delete objects, the process memory pool's size increases and decreases as needed, provided minimum and maximum

values are not yet encountered. This provides dynamic memory allocation of the memory pool.

The RT kernel uses dynamic memory allocation to free unused memory and assign freed memory to other processes. Threads within processes use dynamic memory allocation in the same manner.

For example, some threads periodically need additional memory to improve efficiency such as a thread that allocates large buffers to speed up input and output operations. Such threads can release memory for other threads when they complete, as shown in the next figure.



❶ Threads A and B use memory in the process's memory pool for objects they create.

❷ Thread C completes and then deletes its objects, and releases its memory to the process's memory pool.

❸ Thread D requests memory.

Figure 4-9. Dynamic memory allocation between threads

System calls

Each RT kernel object has an associated set of system calls: programmatic interfaces you use to manipulate objects or control the computer's actions. System calls for one object type cannot manipulate objects of another type. This protects objects from inappropriate actions or changes.

Most system calls have parameters, such as values and names, that you can set to tailor the call's performance. High-level system calls validate these parameters; a condition code returned by the call indicates whether or not the call completed successfully. Invalid parameter usage, such as trying to manipulate an object of the wrong type, or trying to read or write memory to which you have no access, results in the return of a 'call failed' condition code. You can use another system call (GetLastRtError) to determine the failure's exact cause.

Examples of tasks you can perform with system calls include:

Function	System call
Create a new mailbox	CreateRtMailbox
Set a thread's priority	SetRtThreadPriority
Send a handle to an object mailbox	SendRtHandle

For an overview of INtime software's system calls, see *Appendix A, "INtime software system calls"*. For detailed information, including syntax and parameter values, see Help.

Real time shared libraries

You can build libraries that can be shared by one or more real-time applications. These Real time shared libraries (RSLs), function essentially like their Windows DLL counterparts.

An INtime Wizard is provided that integrates with Microsoft Developer Studio to allow you to easily create an RSL. It produces the framework for an RSL, and sets up the various Developer Studio settings to generate an RSL that the INtime loader can manage. For detailed information on creating an RSL, see Help.

Exception handling

While running, an INtime application makes various system calls and interfaces with various hardware devices, the CPU's Numerics Processor, and both self- and system-defined memory structures. Incorrect usage of any of these resources will cause a system exception.

System exceptions include:

System exception	Cause
Programming error	Passing an invalid parameter to a system call.
Environmental error	Requesting resources outside the capabilities/limits of the operating system to grant. For example, a call to malloc can fail with an Environmental exception if the caller's process cannot provide the requested memory.
Numerics exception	Attempting an illegal floating point operation, such as a floating point divide-by-zero operation.
Hardware fault	Attempting to do an operation that violates the protection mechanisms built into the X86 architecture. For example, an integer divide-by-zero operation causes a Divide-by-Zero Fault. Likewise, trying to access a memory address in your VSEG that has not had physical memory assigned to it causes a Page Fault.

Programming and Environmental errors are signaled to threads making system calls as the return value for each call. Application code should check for successful completion of each system call and provide error handling subroutines to handle any possible Programming or Environmental errors.

Proper handling of Numerics exceptions requires that a user exception handler be set up for each thread that makes floating point calls. INtime software provides a Floating Point Exception handler sample project that you can use as a template for your own floating point exception handler.

When a thread causes a Hardware Fault, a system wide Hardware Fault Handler is called. This handler takes one of the following user specified actions:

- Suspends the offending thread (default).
- Deletes the offending thread.
- Deletes the offending thread and its process.

The handler also sends a notification message to the following cataloged data mailboxes:

- HW_FAULT_MBX cataloged in the object directory of the root process.
- HW_FAULT_MBX (if present) cataloged in the object directory of the process whose thread caused the Hardware Fault.

An INtime application can create and catalog a data mailbox in its object directory under the name HW_FAULT_MBX. It can then create a thread that waits at that mailbox for any hardware fault messages and take whatever actions are desired.

INtime software provides a Hardware Fault Exception handler sample project that shows the use of a local HW_FAULT_MBX mailbox and a thread that manages it.

For detailed information on the HW_FAULT_MBX and its notification messages, see Help.

5 Designing RT applications

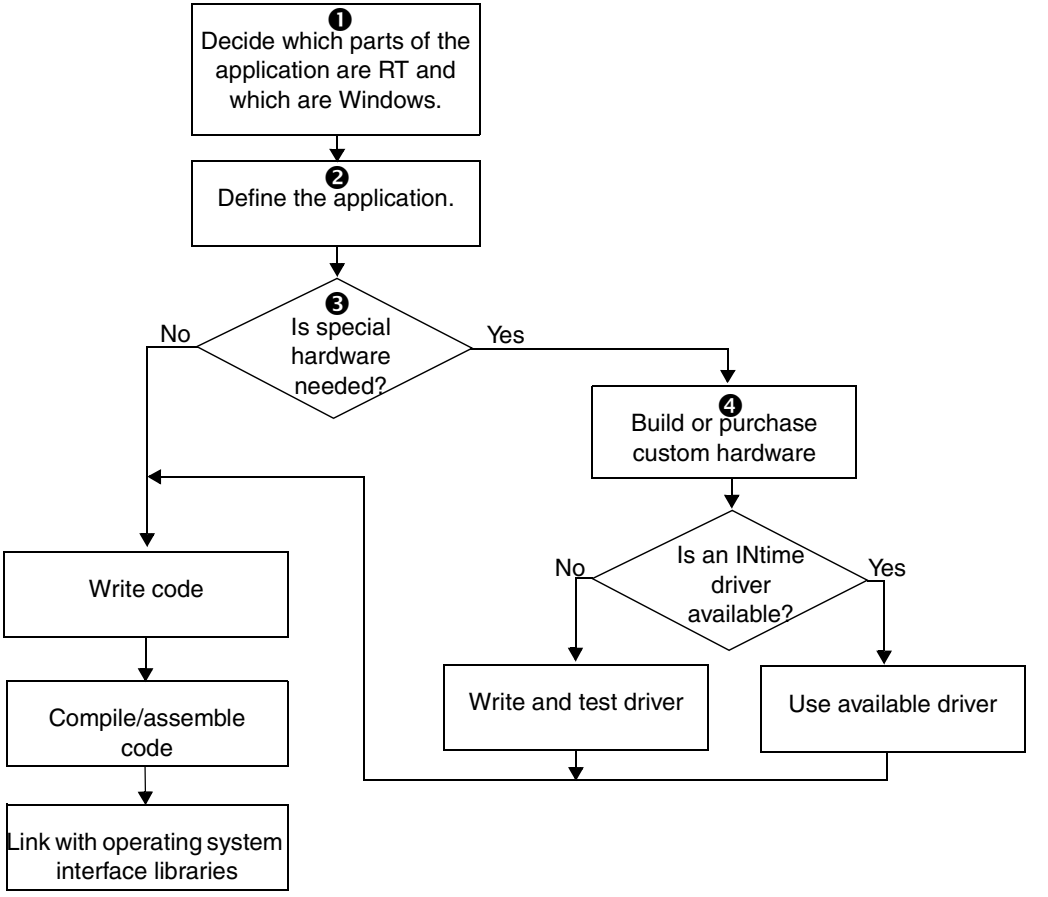
This chapter provides general guidelines for RT system design using INtime software.

Define the application

When designing an RT application, you should include these steps:

- Partition the application into Windows and RT components.
- List the INtime application's inputs and outputs, and decide whether RT or Windows services each system component. Decide which objects to use for inter-thread synchronization and communication, both within the RT portion and between the RT and Windows portions.
- List all threads that require the input and output. Define interrupts and decide which ones require determinism. Assign interrupt levels and priorities to take advantage of multitasking and preemptive, priority-based scheduling.
- Develop the detail for each thread in a block diagram.
- Decide if the application requires multiple processes, and if so, how they will use shared memory and dynamic memory allocation.
- Design your user interface.
- Determine if you require custom devices and drivers.
- Determine if your application will run on a single system or be distributed over multiple RT nodes.

This flowchart shows steps typically taken by RT software designers. When complete, the prototype system is ready to test, debug, and fine-tune.



❶ Partition the application into RT and Windows components.

❷ Define processes, then define threads, interrupt levels, and priorities. Decide which objects to use. Define interrupts, handlers, and levels.

❸ Decide on hardware and device drivers. Decide how to implement a multi-user environment and/or an operator interface.

❹ Decide if you need custom hardware that solves a unique problem or gathers data in a unique way.

Figure 5-1. Typical development cycle for INtime applications

Target environments

Developing INtime applications involves balancing between the target environments:

- The RT environment, where you create portions of the application that require RT robustness and determinism.
- The Windows environment, where you can add RT object access to a Win32 application.

Methodology

A critical aspect of any INtime application is the division of labor between Windows processes and RT processes and, to a finer degree, between the threads in each process.

Important guidelines for developing RT user applications include:

- **Limit RT processes to performing only time-critical functions**: For maximum performance, applications must be divided appropriately between the RT and Windows portions. The INtime software scheduler gives precedence to RT processes and threads. This preferential scheduling guarantees the best possible RT response.

  For example, to optimize user interface (UI) performance, design INtime applications so that RT activity occurs in event-driven bursts. An INtime application that executes for too long on the RT kernel can consume too many CPU cycles and therefore degrade the system's Windows capabilities. This typically causes a sluggish GUI on the Windows side. Adverse effects in the Windows interface, disk, and network operation become noticeable when RT CPU utilization on your system exceeds 80 percent.

- **Determine which Windows threads require the greater relative priority**: The relative priority given to each of the Windows threads of an INtime application can determine the perceived performance of the application in a heavily loaded system. The higher the relative priority given to a Windows thread, the more likely the thread will perform sufficiently in a heavily loaded system.

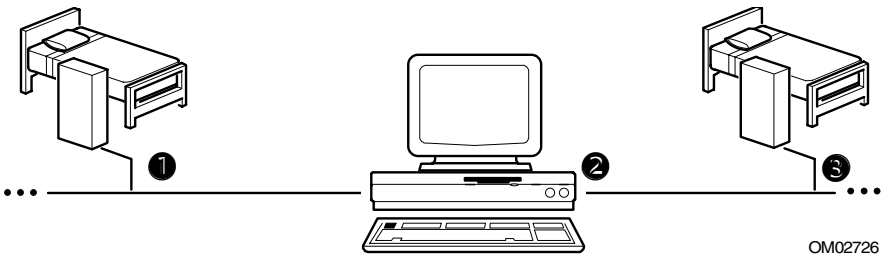
  Determining which threads require the greater relative priority depends on the nature of the application. For example, giving higher relative priority to data manipulation and data storage threads can sacrifice data display and user interface performance when the system is heavily loaded. If an application is data-intensive or the system has no keyboard or display, then sacrificing user interface performance may be desirable. Conversely, if a requirement of an

application is a responsive user interface, then data manipulation and data storage can be given a lower relative priority.

You can use the Windows Performance Monitor to observe CPU usage by Windows and INtime software's RT kernel. For more information, see *Performance monitor* in *Chapter 10, INtime application development*.

A hypothetical system

This hypothetical INtime application monitors and controls dialysis. The application consists of three main hardware components:



❶ A bedside unit is located by each bed. Each unit runs INtime software, which performs these functions:
- Measures the toxins in the blood as it enters the unit
- Adjusts the rate of dialysis
- Removes toxins from the blood
- Generates the bedside display for bedside personnel
- Accepts commands from the bedside personnel
- Sends information to the MCU (Master Control Unit)

❷ The MCU, a PC with a screen and keyboard, runs INtime software. The MCU enables one person to monitor and control the entire system. It performs these functions:
- Accepts commands from the MCU keyboard
- Accepts messages from the bedside units (toxicity levels, bedside commands, emergency signals)
- Creates the display for the MCU screen

❸ A LAN connects the bedside units to the MCU.

Figure 5-2. The hardware of the dialysis application system

The next sections describe how various INtime software features are used in the hypothetical system.

Interrupt and event processing

Interrupts and internal events occur at the bedside units: bedside personnel enter commands asynchronously and the system computes toxicity levels at regular intervals.

Toxicity levels, measured as the blood enters the bedside unit, are not subject to abrupt change. The machine slowly removes toxins while the patient's body, more slowly, puts toxins back in. The result is a steadily declining toxicity level. The bedside units must monitor toxicity levels regularly, but not too frequently. For instance, the bedside units could compute the toxicity levels once every 10 seconds, using a clock for timing. The measurement thread would measure and compute the toxicity, put the information in a mailbox for the MCU, and suspend itself for 10 seconds.

Command interrupts from the bedside unit occur when a medical operator types a command and presses Enter. Interrupts from command entries occur at random times. The interrupt handler signals the interrupt thread. The interrupt thread performs any required processing and waits for the next interrupt.

Processing commands from the bedside units: Each time a medical operator types a command and presses Enter, the bedside unit receives an interrupt signal from the terminal. The bedside unit stops executing the current instruction and begins to execute an interrupt handler.

1. The interrupt handler accumulates the characters in a buffer and puts them in memory. The interrupt handler signals the interrupt thread for bedside commands.
2. The interrupt thread gets the contents of the memory where the handler put the command. It parses the command and does the required processing.
3. The thread puts the command information, along with the number of the bedside unit, into a message.
4. The thread sends the message to the predetermined mailbox for the MCU.
5. The interrupt thread waits for the next interrupt. The system returns to its normal priority-based, preemptive scheduling.

Multi-tasking

Threads in the application run using preemptive, priority-based scheduling. This allows the more important threads, such as those that control the dialysis rate, to preempt lower-priority threads, such as those that update displays. New capabilities can be added to the system by simply adding new threads.

II Using INtime software

This part provides the information you need to install, run, and use INtime software to develop RT applications.

This part contains:

Chapter 6: Installation

Explains how to install and uninstall INtime software.

Chapter 7: Configuration

Describes how to configure INtime software.

Chapter 8: Preparing an RT node

Explains how to set up an RT node to run INtime software.

Chapter 9: Operation

Describes how to start and run INtime software.

Chapter 10: INtime application development

Explains how to use the INtime development environment to create INtime applications.

6 Installation

This chapter explains how to install your INtime software on a Windows system. Follow the directions in this chapter to install INtime software for an INtime node or a Windows node .

For information about preparing RT nodes to run INtime applications, see *Chapter 8, Preparing an RT node.*
For descriptions of INtime nodes, Windows nodes, and RT nodes, see *Topology terminology* in *Chapter 2, Understanding INtime software architecture*.

Install INtime software on a Windows system

Requirements

Installation of INtime software on a Windows NT 4.0 system requires:

- A Pentium† (or better) CPU.
- A minimum of 20MB of RAM.
- A minimum of 20MB of available disk space.
- Windows NT 4.0 with service pack 6 installed and running.

Installation of INtime on a Windows 2000 or Windows XP system requires:

- A Pentium (or better) CPU.
- A minimum of 32MB of RAM.
- A minimum of 20MB of available disk space.
- Windows 2000 or Windows XP installed and running.

The development environment requires the installation of Microsoft Visual C/C++ 6.0.

Before you begin

- Ensure that you are logged on with Administrator privileges.
- Exit all programs prior to installing INtime software.
- If your system has a previously installed version of INtime software, remove it:
  1. Ensure that no INtime services are running. If any are running, be sure they are set to Manual Start (using the Start>Control Panel> Administrative User/Services applet), and then reboot the system.
  2. Select the Add/Remove Programs Applet in the System Control Panel (Select Start>Control Panel>Add/Remove Programs).
  3. Highlight INtime 2.14 (or earlier) program, and then click Remove.

Running the Installation program

To install the INtime software:

1. Insert the INtime software CD-ROM into the CD-ROM drive. The Installation program automatically starts and the welcome screen displays.

   - If you've disabled the automatic start feature on your system, select:
     `Start>Control Panel>Add/Remove Programs`
     Select intime220.msi from the list that displays as you browse the CD.
   - If you are installing on a Windows NTv4.0 system, you may need to install the Microsoft Windows Installer components
     You can do this by executing the instmsiw.exe file found on the INtime software CD-ROM (Start>Run>[CD ROM drive letter]:\instmsiw.exe).

   If the Installation program detects a previous version of INtime software, it prompts you to exit the installation and uninstall the previous version as described in *Before you begin* (above).

   If the HAL currently installed on your machine is not compatible with INtime software, the Installation program prompts you to change the HAL driver. For details, see *About the modified Windows HAL* in *Chapter 2, Understanding INtime software architecture*.

2. Review the INtime License and note its terms and conditions.

   If you continue and install the INtime software, your action shows that you agree to be bound by the terms and conditions outlined in the INtime License.

3. Review the Readme information for any late-breaking information not included in the product documentation.

4. Select a destination directory for the INtime software files, then click the Next button. The default is c:\Program Files\INtime.
5. Select one of the installation options listed below, then follow the prompts in the installation.

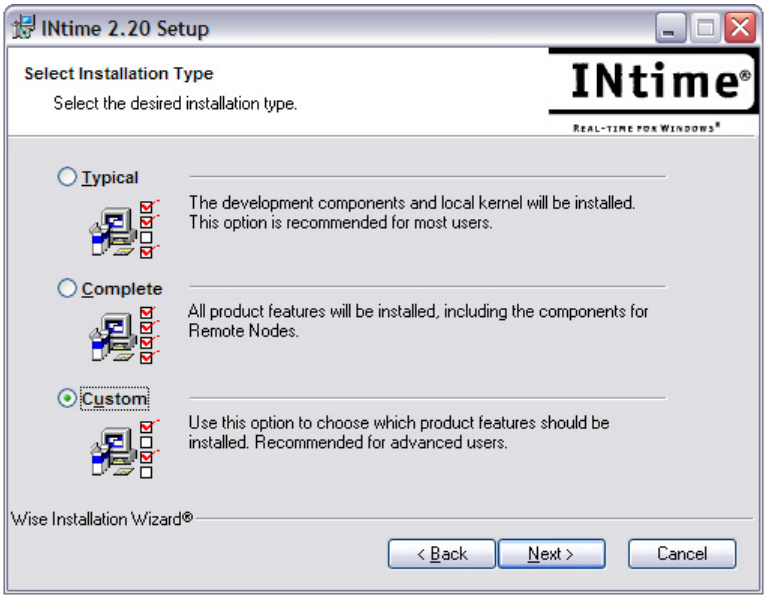

INtime 2.20 Setup
Select Installation Type
Select the desired installation type.
INtime®
REAL-TIME FOR WINDOWS®
Typical
The development components and local kernel will be installed. This option is recommended for most users.
Complete
All product features will be installed, including the components for Remote Nodes.
Custom
Use this option to choose which product features should be installed. Recommended for advanced users.
Wise Installation Wizard®
< Back
Next >
Cancel

Figure 6-1. Installing INtime software

Option	Description
Typical	Installs the INtime Node development and runtime components. Choose this option to develop and run INtime applications on this system.
Full	Installs all product development and runtime components. Choose this option to develop and run INtime applications on this system as well as develop, configure, and run INtime applications on one or more Remote INtime Nodes.
Custom	Installs the product development and runtime feature groups you specify. Choose this option, for example, to make this system a Windows host on which you can develop Remote INtime Node-targeted applications, develop NTX-capable Windows applications, configure Remote INtime Nodes, and communicate with these nodes.

For descriptions of INtime nodes, Windows nodes, and RT nodes, see *Topology terminology* in *Chapter 2, Understanding INtime software architecture*.

6. Click the Next button to install the software.

   When the installation is complete, the Installation program prompts you to reboot the system.

7. Click the Finish button to complete the installation process.

8. Restart your system using one of these methods:

   - Select YES at the last screen. The Installation program restarts your system.
   - Select NO to manually restart your system at a later time.

You must restart your system before running INtime software.

Installing hardware for use with the RT kernel

Before loading an application to interface to a hardware device, it is necessary to prepare the hardware to make it available to the RT kernel.

Under Windows NT 4.0, any hardware will not be controlled by Windows unless a driver is explicitly installed. Under Windows 2000 and Windows XP however the Plug-and-Play software in those systems will attempt to automatically configure hardware it detects at boot time. To prevent this, a Windows driver is provided which allows Windows to believe it has claimed the hardware but also isolates it from other Windows hardware so that an RT application can interact with it.

In an INtime system it is necessary to separate the IRQs of devices that are to be controlled by INtime from those that are to be controlled by Windows. If real-time and Windows devices were allowed to share the same interrupt line, deterministic handling of that interrupt is lost, since release of that signal would depend on the Windows kernel.

To facilitate the separation of interrupts for certain devices, a Windows plug-and-play device driver, called rtdrm.sys, must be installed. This device driver attempts to separate an interrupt line and assign it to a selected device, which may then be used by the INtime kernel. The device driver currently works for PCI devices and Cardbus devices. The driver also then starts the device, making sure that it is powered up.

To install the driver and assign it to a device, perform the following steps:

1. If you are adding your own custom INtime driver for PCI hardware, note the PCI Vendor ID and Device ID numbers for the device.

   Edit the rtdrm.inf file in this directory

   ```
   ...\INtime\system32\drivers
   ```

Add a line to the [MfgDeviceSection] section of the file, being careful that your new line has the next sequence number. For example, a device has vendor ID 0x1234 and device ID 0x5678. The line to add would read:

```
%DeviceDesc53% = INtimeRtdrm, PCI\VEN_1234&DEV_5678
```

2. Next, add a line to the [Strings] section, being careful to match the sequence number of the previously-entered line:

   ```
   DeviceDesc53 = "INtime Acme 8-Channel Coyote Driver"
   ```

   The rtdrm.inf file already contains the required information for INtime PCI drivers provided in the product.

3. Open the Windows Device Manager. To find the device manager, right-click on My Computer, select Manage, then select the Device Manager icon.

4. Select the device you want to allocate to INtime, right-click on it and select the Update Driver... option. Select the “Install from a list or specific location” option, and follow the instructions until you can enter a path to ...\INtime\system32\drivers. Install the driver.

5. If the device is on an interrupt line not physically shared with another Windows device, the driver will install and attempt to start, which may sometimes require a reboot. If it is not possible for Windows to allocate a unique interrupt line, the installation will fail. In this case, you have the option of moving the device to a different PCI slot and try again.

In some systems, the ROM BIOS Setup program can be used to initially group the various Windows devices onto a single IRQ while isolating the realtime devices to unique IRQs. Windows will then start with this information as it makes its IRQ allocations.

In some cases it is not possible to physically isolate the device, in which case the hardware/motherboard combination in your system is not suitable for running the INtime device driver.

To install two real-time devices, the current versions of the Windows plug and play system will attempt to allocate a unique interrupt to each device installed. It might be desirable for multiple INtime devices to share the same interrupt, but currently the plug and play system does not allow that. However, if you can place your device in a PCI slot whose interrupt is already assigned to another RTDRM device, then you can use the Windows Manager to disable your second device as far as Windows is concerned. From the INtime side, however, the device will be visible and its interrupt line will be shared with another INtime device.

To remove all traces of the rtdrm driver from your system:

1. Uninstall all devices assigned to the rtdrm driver. They will all be in the "INtime Rt Device Drivers" group as seen in the Windows Device Manager.
2. Remove from the Registry the \\HKLM\System\CurrentControl-Set\Control\Class\ {EA6EC069-4173-4b17-9C8B-DAD4E734155C} registry entry.
3. Find and delete the oemxx.inf file that is the copy of the rtdrm.inf. The "xx" will be a number starting at "0" and will be in the Winnt\inf (W2K) or Windows\Inf (XP) directory. By doing a search in oem*.inf files for "{EA6EC069", you should find the file to delete. Verify the textual contents before deleteing to insure it's the right file.

7 Configuration

INtime software provides the flexibility to meet a variety of INtime application requirements that you set using configuration options available in the INtime Configuration utility. This chapter describes the following options:

Configuration option	Page
Configuring INtime software	65
Default configuration	65
Running the INtime Configuration Utility	66
Miscellaneous	67
Configuring INtime applications	67
Configuring Windows for non-interactive operation	68
Using the ItWrpsrv.exe Service wrapper	70
Configuring the INtime TCP/IP software	71
Before you begin	71
Hardware installation	71
Setting the TCP/IP configuration parameters	72
NIC driver configuration	72
Advanced configuration parameters	73

Configuring INtime software

Default configuration

By default, the Install program configures INtime software to:

- Run with a keyboard and display.
- Require manual start up for INtime Kernel services. The installation program configures all other INtime services to start automatically.
- Install INtime software files in the `%SystemDrive%\Program Files\INtime` directory.
- Access INtime application wizards, their components, and Help files from the directory appropriate for the version of Microsoft Developer Studio installed on your system.

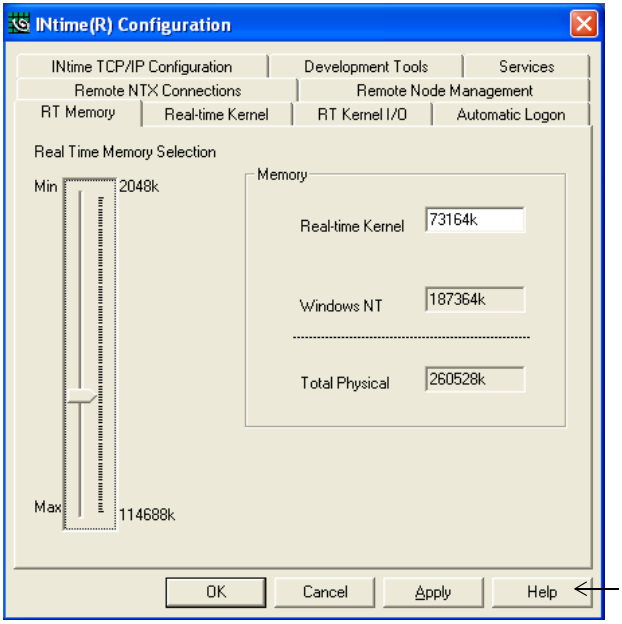
The INtime Configuration utility's Help contains step-by-step instructions for many tasks. You can access this Help by running the utility, then pressing the Help button located at the bottom of the window.

Running the INtime Configuration Utility

To access the Configuration utility, select this program:

```
Start>All Programs>INtime>INtime Configuration Utility.
```

When you start the utility, a tabbed interface displays:



Click this button to view step-by-step instructions fo many configuration tasks.

Tab	Description
RT Memory	Specifies the amount of memory allocated to the RT kernel and displays other memory parameters including total physical memory, and the amount of memory available to Windows.
Real-time Kernel	Configures settings that affect RT kernel operation.
RT Kernel I/O	Configures settings that affect SDM. Also reserves communication ports such as COM1 and LPTR for RT use. You use SDM to disassemble code, set and execute breakpoints, display or change microprocessor registers, and display or change the contents of memory. SDM requires a separate serial debug terminal for its user interface.
Development Tools	Installs, uninstalls, and verifies the status of INtime wizards for Microsoft Developer Studio. **Note**: You must have the appropriate version of Microsoft Developer Studio installed before you can install the INtime wizards.

Tab	Description
Automatic Logon	Configures Windows and INtime software for non-interactive operation, including operation without a keyboard or mouse.
Services	Configures INtime Services for either automatic or manual startup. **Note**: You can also configure these services in the Windows Service Control Manager. Displays current status of each INtime service and provides buttons to start non-running services.
Remote NTX Connections	Configures the connection between a Windows host and its RT nodes.
Remote Node Management	Edits configuration files on remote nodes. It also provides basic file system access to RT nodes including: directory creation and deletion, copying files to and from remote nodes, disk free space on remote nodes, and System Info about remote nodes.
INtime TCP/IP Configuration	Sets the parameters for the RT Network Stack on an INtime node.

Miscellaneous

RTIF.SYS driver

When you install INtime software as an INtime node, the installation process sets up the Windows system to allow RtIf.sys, the RT kernel mode device driver, to load at boot time.

To disable the driver from running, use the Windows Device Manager to change the startup mode of the RtIf.sys driver from System to Disabled.

Interrupt resources

By default, INtime software takes over the system clock (IRQ0) and multiplexes the clock interrupt between the RT and the Windows kernels. INtime applications can take over other interrupts, but must ensure that Windows drivers do not try to use the associated devices.

Configuring INtime applications

This section details the steps required to configure INtime applications for specific conditions. You can also access this information in the INtime Configuration's Help file.

Configuring Windows for non-interactive operation

Both INtime software and Windows can run without an attached keyboard or mouse. To run INtime software without a keyboard and mouse:

1. Verify that the PC's Basic Input Output System (BIOS) executes properly without a keyboard and mouse.
2. Configure all INtime software services to execute automatically during Windows restart.
3. Shut down Windows.

Verifying BIOS execution

Executing Windows and INtime software without an attached keyboard and mouse requires that the PC's BIOS execute to completion when a keyboard and mouse do not exist. To test whether a PC supports execution without a mouse and keyboard:

1. Configure the Windows system so that it is visible on a network, and then enable remote shutdown of the system under test.
2. Shut down Windows using one of these methods:
   - Select the Start menu's Shutdown option.
   - Depress the Ctrl+Alt+Del keys and then click the Shut Down button.
   - Use the remote shutdown mechanism (see step 1).
3. Turn the PC's power off.
4. Unplug the keyboard and mouse from the PC.
5. Restore power to the PC.

   If Windows successfully restarts, then the PC's BIOS executed to completion and you can use the PC without a keyboard and mouse.
6. Shut down Windows via the remote shutdown mechanism.
7. Remove power from the PC.
8. Plug the keyboard and mouse back into the PC.
9. Restore power to the PC.

Configuring Windows for non-interactive logon

Executing Windows and INtime software without an attached keyboard and mouse requires that Windows be configured for non-interactive logon, also known as AutoAdmin Logon. To configure Windows for non-interactive logon:

1. Log on to Windows with Administrator privileges.
2. Create/verify the presence of a Windows user with a password as to which you want to automatically log on.
3. Start the INtime Configuration application
4. Select the Automatic Logon tab, then edit these fields:
   - **Automatic Logon Options**: Select Auto Logon Always.
   - **Auto Logon Parameters**: Enter the information in these fields:

     Domain Name
     User Name
     Password

     Note: Be sure to use the User name and Password established in step 2..
5. Exit the INtime Configuration application.
6. Shut down Windows using one of these methods:
   - Select the Start menu's Shutdown option.
   - Depress the Ctrl+Alt+Del keys and then click the Shut Down button.
   - Use the remote shutdown mechanism.
7. Restart Windows.

When Windows restarts, it automatically logs on as the specified user in the specified domain.

Deploying a Windows system with the non-interactive logon feature enabled represents a potential security breach: the Windows registry stores the specified user, domain, and password in clear text–they are not encrypted.

Configuring INtime software services to execute automatically

Executing Windows and INtime software without an attached keyboard and mouse means that all INtime software services required by an INtime application must start automatically during a Windows restart.

To configure an INtime software service to start automatically during Windows restart:

1. Go to the Services tab.
2. Choose Start Service Automatically for each INtime service you require:
   - INtime Kernel
   - INtime Remote Connection
   - INtime Node Detection
   - INtime I/O
   - INtime Event Log
   - INtime Registry
   - INtime Clock Synchronization
   - INtime Network
3. Click the OK button
4. Restart your system. “Start Services” changes made on this tab take effect only after you reboot the system.

Using the ItWrpsrv.exe Service wrapper

INtime software includes the Service wrapper that you can use to install or run any .EXE or .RTA file as a Windows service. You can then designate the service to start at system boot. You must have System Administrator privileges to use this wrapper.

The Service wrapper installs itself as the binary for the specified service. When invoked by the Services applet, the wrapper uses Win32’s CreateProcess call to start the executable associated with this instance of the Service wrapper.

The Service wrapper includes the ItWrpSrv.exe file. For detailed information about how to use the Service Wrapper, select this INtime Help option:

```
Using INtime software>Services>Service wrapper
```

For information about accessing Help, see *About INtime software* in the *Before you begin* chapter at the beginning of this guide.

Configuring a Windows service to execute automatically

To configure a Windows service to start automatically during system reboot:

1. Activate the Services applet found in Administrative Tools on the Control Panel (Start>Control Panel>Administrative Tools>Services).
2. Highlight the service you want to start automatically.
3. Click the right mouse button.
4. Select Properties.
5. Select Automatic as the Startup type.
6. Click the OK button.
7. Restart your system. Changes made take effect only after you reboot the system.

Configuring the INtime TCP/IP software

To configure the INtime kernel TCP/IP software, use the INtime TCP/IP Configuration tab in the INtime Configuration Utility.

Before you begin

You will have to assign an IP address and host name to the network interface controlled by the INtime TCP/IP software. This should not be the same address used by your Windows networking software.

Hardware installation

Before configuring INtime TCP/IP software, install and configure the network interface adapter you plan to use. Instructions for installing hardware for RT kernel use may be found in *Installing hardware for use with the RT kernel* in *Chapter 6, Installation*.

Setting the TCP/IP configuration parameters

1. Start the INtime Configuration Utility.
2. Select the INtime TCP/IP Configuration Tab.
3. Set the following:

To...	Select this checkbox...
Enable real-time networking	Enable Real-time Networking
Automatically start real-time networking services	Auto-Start Networking Services

4. Select the Transport Layers you plan to use.
5. Enter the host name.
6. Click the NIC Configuration button. The NIC Configuration Dialog displays. Edit or Add the interfaces you require, as needed. Each configured interface needs these items:

   IP address
   IP interface address mask
   Gateway address
   NIC driver with its associated parameters.

7. Exit the NIC Configuration Dialog.
8. If you want to use a DNS server for name resolution, select the DNS checkbox, then enter the domain name and up to three IP addresses of DNS servers.
9. Click the Apply button.

If the kernel is already loaded, you must restart the system for changes to take effect.

NIC driver configuration

When you add or edit a NIC interface, you will need to select the NIC driver appropriate to your hardware, and configure the parameters appropriate to your hardware. Normally an ISA card requires configuring of the IRQ and Base Address. PCI cards do not normally require any special configuration parameters.

However, if there are multiple PCI NIC cards in the system of the same type (for example, two Intel EtherExpressPro/100 cards), then you must specify the proper instance number for the card you want to manage with the specified INtime driver.

Advanced configuration parameters

In some instances it may be necessary to change default settings for the real-time networking services (e.g. IP, TCP, or UDP). You can configure these parameters by clicking on the Advanced button in the NIC Configuration Dialog. The Advanced Settings Dialog appears which allows you to choose the section you which to configure. Once you choose a section, the parameters for that section are displayed with their values. You can edit these values or add your own.

Static routes and aliases may be added to the configuration using the INtime Configuration utility. See the Configuration help file for further details.

8 Preparing an RT node

This chapter explains how to set up an RT node to run INtime software.

Topic	Page
Requirements	75
Configuring an RT subsystem	76
Building remote node boot/installation floppies	77
Loading a user application on an RT node at boot time	79
RtLoad.xxx file format and switches	80
First character	80
First space	80
Editing the RtLoad file in-line	80

Requirements

Installing INtime software on systems used as RT nodes requires:

- A PC that contains an Intel386† EX or better microprocessor.
- A minimum 4MB of DRAM for the debug version of the RT kernel; 2MB of DRAM for the non-debug version of the RT kernel.
- PC-compatible local media. (Where the BIOS uses INT13).
- RT node files:

Files/directories	Description
bluecon.rta	An INtime application that manages the system console.
eepro100.rta	An Intel EtherExpress† Pro/100 NIC driver.
ping.rta	An RT application that tests the network connection between two machines.
ne.rta	An INtime driver for the NE2000 compatible NIC (ISA card).
3c59x.rta	A 3COM 3C59XX NIC driver.
rtl8139.rta	A Realtek rtl8139 NIC driver.
ntxproxy.rta	An INtime application that manages NTX calls on the RT node.
rrtimage	An RT kernel binary image without debug capabilities for use on an RT node.
rrtimage.dbg	An RT kernel binary image with debug capabilities for use on an RT node.

rtcimcom.rta	An INtime application that manages the serial NTX communications channel.
rtcimudp.rta	An INtime application that manages the UDP NTX communications channel.
serdrvr.rta	An INtime driver for the serial devices on the remote RT node.
loopback.rta	An INtime loopback driver for use with the TCP/IP stack layers.
ip.rta, rip.rta, udp.rta, tcp.rta	TCP/IP stack layers.
rtconfig.sys	A file generated by the Configuration Utility that sets up boot-time configuration parameters for the remote RT kernel.
rtboot.ini	A file generated by the Configuration Utility that sets up boot-time configuration parameters for the remote INtime bootloader.
rtload.xxx	A file generated by the Configuration Utility that specifies what files to load at boot time on the RT node. The extension xxx can either be udp if the UDP/IP NTX channel is used, or ser if the Serial NTX channel is used.

- bldflpys.bat, A file generated by the Configuration Utility that is used to create installation floppies used to install the Remote INtime files onto an RT node's Hard Disk or Flash system, or for use in booting the RT node from the floppy disks themselves.
- RtSetup.exe, a DOS program used to make an RT node's boot device bootable. This file is located in either the `\INtime\remote\common` directory or on the INtime RT node setup disk.

Configuring an RT subsystem

To build an RT subsystem:

1. Ensure that INtime software is installed on the Windows system you want to use as the Windows host for your RT nodes.
2. Determine the type of RT subsystem you want to build. Subsystem types are categorized by the type of Boot Media on the remote system:
   - Floppy diskette
   - Hard disk
   - Flash disk
3. Build the RT subsystem floppies. Differences in the directions based on the subsystem type you select will be noted.

Building remote node boot/installation floppies

1. Start the INtime Configuration utility and select the Remote NTX Connections tab.
2. Click the Add button in the Remote NTX connections tab. The Remote Client Settings screen displays.
3. Complete the fields on each of these tabs:
   - NTX tab

     Be sure to select the appropriate Remote Client Channel and complete the prompts for that channel that display on the right side of the screen.

   - RT Kernel tab
   - Loader tab
   - Boot tab
   - Setup tab

     If the RT node's boot media is a set of floppy disks, select "Floppy"' as the Boot Media.
     If the RT node's boot media is a Hard Disk, select "Harddisk" as the Boot Media.
     If the RT node's boot media is a Flash Disk, select "Flashdisk" as the Boot Media.
4. Click the OK button. The Remote NTX Connections tab displays.
5. Click the Apply button to update the registry with information about this RT node.
6. Click the Build System button to build the RT subsystem files with the settings you specified. If a message indicates that a setting is not correct, you must fix the error, click the Apply button, then build again.

   When successful, a message tells you where the build files were placed.
7. Build RT node installation/boot floppy disks:
   A. Double-click the bldflpys.bat file and follow the directions when prompted. This file builds two floppy disks used to boot/install the RT subsystem.
   B. Ensure that no error messages display while the batch file builds the floppies and name (label) the floppy disks when prompted during each FORMAT operation.

C. Label the floppies as follows:

If you selected "Floppy" as the Boot Media in step 3 (on page 77), label the two floppy disks as follows:

Floppy 1: Boot floppy
Floppy 2: Commands floppy

If you selected any other option for Boot Media in step 3, label the two floppy disks as follows:

Floppy 1: Installation Floppy 1
Floppy 2: Installation Floppy 2

Booting from the Boot floppy disk

If you selected "Floppy" as the Boot Media in step 3 (on page 77):

1. Place the Boot floppy in the RT node's floppy drive.
2. Press Ctrl-Alt-Delete. The system reboots, then loads INtime software's remote RT kernel from Boot floppy.
3. Insert the Commands floppy when prompted to do so. The system loads the system applications from the Commands floppy. After the applications load, the RT Node can communicate with the Windows host.

Copy build files to the hard/flash disk

If you selected "Harddisk" or "Flashdisk"as the Boot Media in step 3 (on page 77):

1. Place the INtime RT node setup disk in the target RT node's floppy drive and boot. The DOS prompt displays.
2. Partition the hard/flash disk:
   A. Invoke the 'fdisk' command. A character-based GUI displays.
   B. Ensure that partition 1 is available for use.
   C. Select the option that creates DOS partition 1.
   D. Specify the amount of memory allocated to the primary DOS partition.

   The RT kernel can only detect DOS partition 1.

   E. Press the Esc key to exit fdisk and reboot the system. Leave the RT node setup disk in the drive.
3. Format the hard/flash disk by entering:

```
format c: /u
```

4. Run the RT setup program by entering:

   ```
   rtsetup c: 1
   ```

   where `c:` is the drive and `1` is the partition number.

5. Copy the contents of Installation Floppy 1 and Installation Floppy 2 (created earlier) to the hard/flash disk:

   A. Insert Installation Floppy 1, prepared by the batch file, into the floppy drive and then enter this command to copy the floppy disk's contents to the hard/flash disk:

      ```
      copy *.* c:
      ```

   B. Insert Installation Floppy 2, prepared by the batch file, into the floppy drive and then enter this command to copy the floppy disk's contents to the hard/flash disk:

      ```
      copy *.* c:
      ```

   C. Remove the floppy disk and reboot the system.

      The system reboots and loads the INtime software's remote RT kernel and system applications. The RT node can now communicate with the Windows host.

Special requirements for Flash disks

- Ensure that DOS can detect the flash disk as the C: drive.
- Ensure that you can format and fdisk the flash disk from DOS.
- Ensure that these basic ROM BIOS functions are available:
  - INT 13H (disk I/O)
  - INT 15H (Extended memory access)
  - INT 10H (video)

Loading a user application on an RT node at boot time

To have one or more user applications load at boot time on an RT node, simply add them to the appropriate RtLoad file after the system applications. For example, to cause the application MyApp.rta to load with input parameters `"base=0x760 irq=5"` on an RT node set up to boot up with a UDP-based NTX interface to its Windows host, modify the RtLoad.udp file as follows:

- After `ntxproxy.rta`, add this line:

  ```
  \myapp.rta "base=0x760 irq=5"
  ```

- Set process-level parameters in RtLoad.sys following the switch setting information described in the next section.

RtLoad.xxx file format and switches

The RtLoad.xxx file is generated by the INtime Configuration utility, based on user input. As the RT image boots, it finds the RtLoad file specified in the RtConfig.sys configuration file and parses its contents to determine which applications to load, and in what order. Each line (terminated with a LF character) in the file can indicate the full pathname of an application to load as the RT system initializes. Applications load in the same order as they appear in the file. The load file has the following per-line format:

First character

Mode	Description
;	Treat the entire line as a comment.
#	Print the rest of the line on the current STDOUT device and wait for a response from the STDIN device before proceeding to the next line. This is useful when booting from a floppy device and then pausing to insert the second (or nth) floppy of the set before proceeding. (Only supported if Logging Mode is set to “Stdout” and Printf/Scanf console is set to “Local” on the Loader page of the INtime Configuration utility’s Remote Client Settings tab.
\	Indicates that the following characters before the next space represent an application program in the root directory of the boot device.
^	Processes the following character as a command to the RIRTAL (Remote INtime Run-Time Application Loader). Supported command characters include: E Tells the loader to enlarge the code segment (same as Open Data Segment in the INtime Loader) for the next application listed for loading in the RTLOAD.XXX file.
Other	Any other character is assumed to be the first letter of the application program assumed to be in the root directory of the boot device.

First space

In all lines except those that begin with ‘;’, ‘#’, or “^”, the first space character is interpreted as the delimiter of the application file pathname to be loaded. The rest of the line’s characters are of two types:

- Parameters passed to the RIRTAL to initially load the file.
- A string passed to the loaded application for its examination via the ‘argc’ and ‘argv’ C-library commands.

The RIRTAL-load parameters are all prefixed with a ‘-‘ (dash) and the application parameters are contained within a quoted string.

The following RIRTAL-targeted load parameters are as follows:

Parameter	Description
`-o`	Object directory size (number of entries) of the application being loaded. If not specified, the EIOS default I/O job object directory is used. If not a multiple of 16, the specified value is rounded down to the nearest multiple of 16.
`-n`	Minimum size of the application's memory pool in bytes. If not specified, the minimum pool size is calculated from information found in the binary of the application being loaded.
`-x`	Maximum size of the application's memory pool in bytes. If not specified, the maximum pool size will be 0xffffff.
`-v`	Size in megabytes of the VSEG created for Ring 3, Flat applications. If not specified, a default value of 8 megabytes will be used. A value of 0 or 1 causes the Default VSEG_SIZE value specified in the ICU to be used. Values above 512 are reduced to 512. The value specified is multiplied by 4 Mbytes. **Note**: the VSEG size specified here must be large enough to hold the amount of physical memory needed by the application.
`-p`	Priority of the initial thread of the application being loaded. If not specified, a default of 136 is used.
`-w`	Number of seconds the Loader should wait before loading the next application specified in the load file.
`-d`	Tells the RIRTAL to place an Int 3 instruction at the first instruction of the loaded application so that the system breaks to the monitor when the application begins to run. This is useful for debugging.

Any parameters other than those listed above cause the line to be ignored.

The argument string within a pair of " (double quote) characters can be up to 256 characters in length. It is passed intact. It cannot contain any " (double quote) characters itself. Both the beginning double quote and the ending double quote must be found on the same line, otherwise the line is ignored. Valid load parameters can appear after the argument string on a given line.

A SPACE character must precede both load parameters and argument strings. No SPACE character is allowed between a load parameter ID and the parameter itself. For example, `testfile -o128 "This is a test"` is a valid entry that causes the file testfile to load with an object directory size of 128, and with the string "This is a test" passed to it as an argument. Likewise, the example `testfile1 -v 16 "This is a test"` is invalid since:

1. There is a SPACE between the 'v' and the 16, and
2. There is no SPACE preceding the argument string "This is a test".

Editing the RtLoad file in-line

Once the Remote node is configured and started, you may need to edit the RtLoad.xxx file to, for example, add or delete applications loaded at startup. You can use the INtime Configuration utility to edit this file:

1. Open the INtime Configuration utility and select the Remote Node Management Utility tab.
2. Click the Nodes button and choose the remote client node that you want to configure. Once an active node is selected, the root directory displays in the pane.
3. Navigate to the file you want to edit. You can navigate through the directory structure by double-clicking into folders and by pressing the "Up One Level" button. Highlight the text file you want to edit.
4. Click the "Edit Remote File" button. The utility copies the file to the host system and opens the file in an editor. Be sure to save any desired changes and exit the editor when you are finished.
5. Click the OK button in the dialog box to copy the updated file to the remote node.

You can also use the utility to transfer files to and from the remote node, and to restart the remote node.

9 Operation

This chapter describes how to start and operate INtime software.

You can treat the RT kernel and associated INtime applications as Windows services and start them automatically when Windows initializes.

In the event of a Windows blue screen crash, INtime software keeps running until a graceful shutdown occurs. To start INtime software again, you must first restart Windows.

Starting the RT kernel and related components

1. Open the Windows Service Manager (Start>Control Panel>Administrative Tools>Services).

   The Windows Services table lists these INtime services:

Table 9-1. INtime software's Windows services

Service	Description
INtime Kernel Loader	Loads the RT kernel binary image into memory and starts it. Uses loadrtk.exe.
INtime Network service	A Windows service which loads the TCP/IP network processes onto the INtime kernel, according to the configuration defined by the INtime Configuration utility.
INtime ClockSynchronization service	A Windows program that synchronizes RT client time-of-day clocks with the Windows host's time-of-day clock.
INtime Event Log service	A Windows program that acts as a gateway to allow RT applications to log events in the system event log on the Windows host.
INtime I/O service	A Windows program that acts as a server to RT "C" library to obtain a console window for display (via printf) of application data and to receive (via scanf) user input from the system keyboard. This service also provides Windows file system support to the RT "C" library.

Table 9-1. INtime software's Windows services

Service	Description
INtime Node Detection service	A Windows program that detects RT clients, both local and remote. This program checks for and registers RT clients that exist in both of these locations: • RT clients configured in INtimeConfiguration.exe. • RT clients available to the system.
Intime Remote Connection manager	A Windows program that detects and manages connections between the Windows host and its RT nodes. The manager includes NtxRemote2.exe, which is required for a Windows host to communicate with RT nodes.
INtime Registry service	A Windows program that provides Windows registry access to RT processes via the RT application library.

2. Start each INtime software service as desired. By default all services except the INtime Kernel and INtime Network services start automatically. You can change this in the INtime Configuration utility:

   A. Open the utility and select the Services tab.

   B. Start INtime services here, configuring them to start either automatically or manually.

You can perform the same functions from the Windows Service Manager.

INtime service dependencies

Some INtime services require other INtime services. If you start an INtime service that has such a dependency before starting the services it requires, that service automatically starts the services it requires.

This figure shows the INtime service dependency hierarchy::

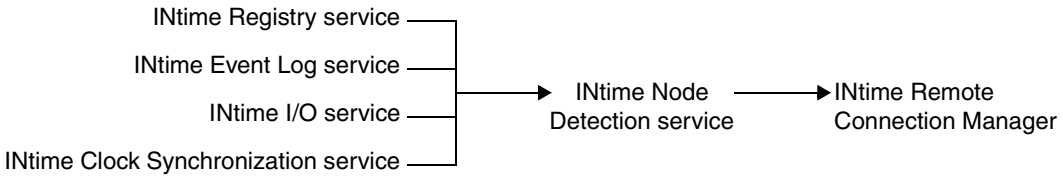


Figure 9-1. INtime service dependency hierarchy

After you start INtime software

- Use Microsoft Developer Studio to develop your INtime application.
- Use the Soft-Scope debugger to load and debug your RT applications. You can start this program by selecting Start>All Programs>INtime>Soft-Scope Real-time Application Debugger.
- Use the INtime Loader to load and run your RT applications. You can start this program by selecting Start>All Programs>INtime>RT Application Loader.
- INtime tools available from the Start>All Programs>INtime software menu include:

One or more of these tools are installed, depending on which INtime Environment option you selected when you installed INtime software.

Tool	Description
INtime Configuration	A Windows program that configures INtime software.
RT Application Loader	A Windows program that loads/starts the RT portion of INtime applications.
Soft-Scope Debugger	A Windows program that communicates using NTX calls, with the INtime debug server (SSKernel) to provide dynamic source level, multi-tasking debug capabilities for RT applications. **Note**: For detailed information about using Soft-Scope, see Soft-Scope's Help.
Starvation Alarm	A Windows program that detects starvation based on how many CPU cycles Windows receives within a specified time interval.
INtime Explorer	A Windows program that uses NTX calls to communicate with its self-loaded RT counterpart to display objects inside an INtime node.
INscope Real-time Performance Analyzer	A Windows program that uses NTX calls to communicate with its self-loaded RT counterpart to trace execution of INtime applications.
RT Process Manager	An RT application that displays the currently active RT processes in the system. It also queries the user for processes to delete.

Also available from the Start>All Programs>INtime software menu are these resources:

- **INtime Help**: Describes INtime software concepts and explains how to use INtime tools. Help includes all system calls, including their syntax which you can cut and paste directly into your code.
- **INtime Release Notes**: Lists features and issues that arose too late to include in other documentation.
- **Dinkum EC++ Help**: Describes the Dinkumware EC++ software included with the INtime product.
- **Sample Code and Projects**: Menu of MSVC projects that demonstrate various aspects of INtime application programming.

10 INtime application development

This chapter describes how to create INtime applications.

Typically, you develop both Windows and RT source code as shown in this illustration. The remainder of this chapter details each of these steps.

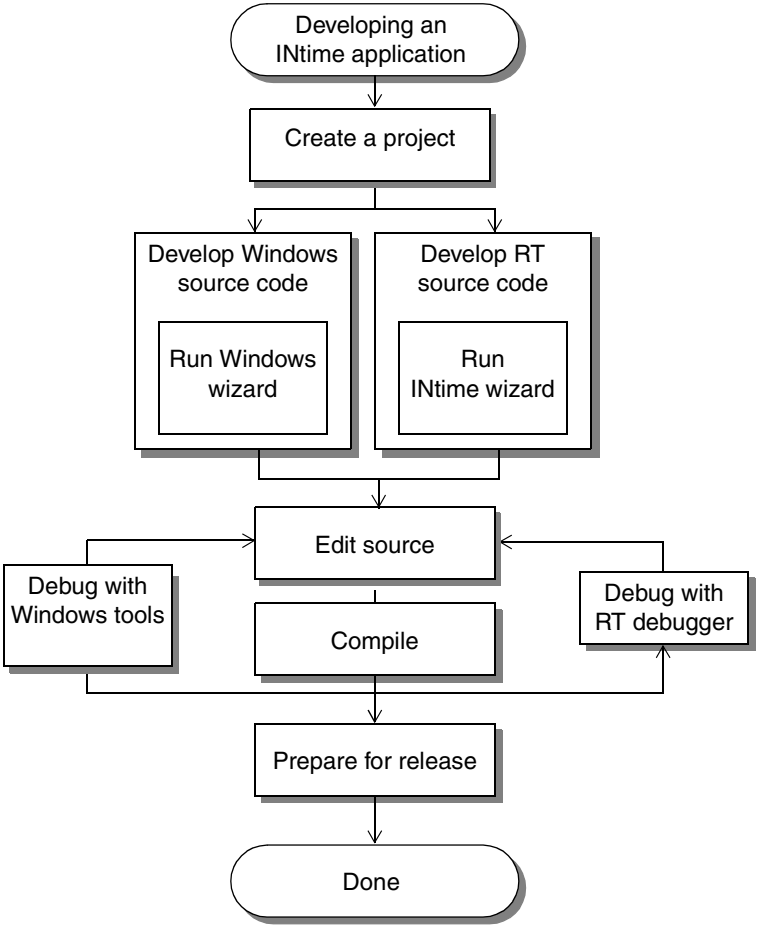


Figure 10-1. Developing an INtime application

Create a project

To develop RT applications using INtime software, you must have INtime installed on your system, and Microsoft Developer Studio (version 6.0) running.

Before creating the project, decide how to set it up. Typical structures for INtime applications include:

- Set up the Windows portion as a project, and the RT portion as a sub-project. Use this approach when you want to start your INtime application in Windows and invoke the RT functions.
- Set up the RT portion as a project, and the Windows portion as a sub-project.
- Set up each portion as a project.

Keep in mind that an INtime application results in a minimum of one Windows executable (a .DLL or .EXE file) and one RT executable (an .RTA file).

Develop Windows source code

To develop the Windows portion of an INtime application, use Microsoft Developer Studio as you normally would:

1. With Microsoft Developer Studio running, select a standard application wizard. Use the wizard to build the Windows portion of your INtime application.
2. Use the Microsoft Developer Studio's editor to edit the generated code and link the application.
3. When editing your Windows source code, use the NTX calls provided with INtime software to access RT functionality.
4. Debug with the debugger included as part of Microsoft Developer Studio.

Adding the INtime RT Client Browser to your INtime application

The INtime RT Client Browser (INBROW.OCX) is an ActiveX control that you can add to your INtime applications. Add the browser by selecting Project/Add to Project/Components and Controls on the Microsoft Developer Studio menu. Select Registered ActiveX Controls/INtime Node Browser and add it to your project. The Node Browser then appears as an available control on the Controls Toolbar. Add the control to the dialog you desire. For information about including ActiveX controls in projects, see the Microsoft Developer Studio documentation.

Once in your project, the control offers these functions:

Function	Description
GetCurrentName	Obtains the name of the item highlighted in the INtime RT Client Browser.
GetCurrentState	Obtains the state of the item highlighted in the INtime RT Client Browser. Valid states include: 0 Not an INtime node 1 ACTIVE 2 OFFLINE 4 CONFIG Where: ACTIVE RT nodes the browser currently can communicate with. OFFLINE RT nodes the browser has but cannot currently communicate with. CONFIG RT nodes the browser has not communicated with. These nodes may not exist.
GetMask	Indicates which RT nodes display in the INtime RT Client Browser.
SetCurrentName	Not supported.
SetCurrentState	Not supported.
SetMask	Displays in the INtime RT Client Browser only the nodes with the state(s) you specify: 0 Not an INtime RT node 1 ACTIVE 2 OFFLINE 3 ACTIVE + OFFLINE 4 CONFIG 5 CONFIG + ACTIVE 6 CONFIG + OFFLINE 7 CONFIG + OFFLINE + ACTIVE Where: ACTIVE RT nodes the browser currently can communicate with. OFFLINE RT nodes the browser has but cannot currently communicate with. CONFIG RT nodes the browser has not communicated with. These nodes may not exist.
SetSelected	Identifies which INtime RT node to select in the browser window when you specify a pointer to a string that contains a node name. This function returns TRUE if the name is valid and selected; otherwise it returns FALSE.

Develop RT source code

To develop the RT portion of an INtime application, use the INtime wizards available in Microsoft Developer Studio. The INtime wizards guide you through the decisions required to develop the RT portion of an INtime application and generate the corresponding code framework which you fine-tune in the Microsoft Developer Studio's editor.

You can select one of these INtime wizards:

- **RT process wizard**: develops the RT portion of RT applications.
- **RT Process Add-in wizard**: adds supplemental files to the already-generated RT portion of INtime applications. This wizard supports only code written in EC++.
- **RT device driver wizard**: develops RT drivers.

The RT portion of INtime applications support only source code written in C or EC++. They do not support applications written using MFC.

- **RT RSL Wizard**: generates code for a Realtime Shared Library.

Running the INtime RT process wizard

To create the RT portion of an INtime application:

1. Select what you want to generate:
   - An empty project (the wizard simply sets up the correct compiler and linker settings for you). If you select this option, go to step 4.
   - A simple "Hello World" application (the wizard creates a simple source file and adds it to the project). If you select this option, go to step 4.
   - A full-featured INtime application. If you select this option, continue to the next step.
2. Add elements to your process:
   A. Select one or more elements from the main screen to add to your RT application:
      - Mailbox or semaphore server thread
      - Thread that operates at a regular interval
      - Interrupt handling
      - Shared memory allocation

- Client thread

For detailed information about the fields on these screens, see Help. For information about accessing help, see *Where to get more information* on page v.

B. Click the Add element button. That element's detail screen displays.

C. Specify element parameter values.

D. When satisfied with an element's settings, click the OK button. The wizard's main screen displays again.

3. (Optional) Change the global process settings:

   A. Select -global- option from the main screen.

   Note: You must click in the Name column to access this detail screen.

   B. Specify global process values.

   C. When satisfied with the global settings, click the OK button. The wizard's main screen displays again.

   Note: If you don't access this screen now, the wizard will prompt you to verify global settings before generating the process.

4. Generate the process. The wizard creates a project from the settings you specified.

   A. Click the Generate Process button. If you did not access the Global Settings screen, the wizard prompts you to edit or accept the global process settings. The wizard then displays the New Project Information screen and prompts you to verify the process information.

   Note: You must click in the Name column to access this detail screen.

   B. Click the OK button. The wizard creates a process from the settings you specified. A process includes, at a minimum, these files:

   - Main file (which has the project name) with a .C or .CPP extension; for example: TEST.C.
   - Project file TEST.DSP and TEST.DSW.
   - Project header file with a .H extension.
   - Utility function file, UTIL.C.
   - A C source file with a .C extension for each thread.
   - A text file called README.TXT that describes each generated file.

Running the INtime RT process add-in wizard

To modify an existing RT process:

1. Add elements to your process:
   A. Select one or more elements from the main screen to add to your RT application:
      - Mailbox or semaphore server thread
      - Thread that operates at a regular interval
      - Interrupt handling
      - Shared memory allocation
      - Client thread

      For detailed information about the fields on these screens, see Help. For information about accessing help, see *Where to get more information* on page v.

   B. Click the Add element button. That element's detail screen displays
   C. Specify element parameter values.
   D. When satisfied with an element's settings, click the OK button. The wizard's main screen displays again and a message indicates what the wizard added to that process.
   E. When satisfied with the global settings, click the OK button. The wizard's main screen displays again.
2. When you are satisfied with your additions, click the Close button to exit the wizard.

Running the INtime RT device driver wizard

To create an RT device driver:

1. Specify parameter values on the screen that displays.
2. When satisfied with an element's settings, click the Next button. The wizard's next screen displays.
3. Repeat steps 1 and 2 for each of these screens:
   - General service information
   - Options for the service
   - Sizes of information for the service
   - Port number ranges for the service

The last screen is one of these:

- Implementation details for polling
- Implementation details for interrupt handling
- Implementation details for port-based event
- Implementation details for custom event

- The last screen that displays depends on the information you provided on the first screen.
- For detailed information about the fields on these screens, see Help. For information about accessing help, see *Where to get more information* on page v.

4. Click the Finish button. The wizard prompts you to verify new project information.
5. Click OK. The wizard creates a project from the settings you specified. A project includes these files:

- Main file (which has the project name) with a .C extension; for example: TEST.C.
- Project file TEST.DSP and TEST.DSW.
- Project header file with a .H extension.
- Utility function file, UTIL.C.
- Implementation file, IMPL.C
- A text file called README.TXT that describes each generated file.

Running the INtime RSL wizard

The INtime RSL (Real-time Shared Library) wizard generates the framework for a Real-time Shared Library. The code generated illustrates how to export both function and variable names, and also initializes the project settings to correctly generate an RSL.

The generated files include:

- Main file (which has the project name) with either a .C or a .CPP extension, depending on which option was chosen, for example TEST.C.
- Project files, for example TEST.DSP and TEST.DSW.
- Project header file, for example TEST.H.
- Utility function file, UTIL.C.

The main file contains a function RslMain which you can modify, if required. A default RslMain function is linked if this function is deleted from the source file. For further details about RSLs, see Help.

Compile

Use Microsoft Developer Studio to compile and link the application. The RT portion of INtime applications requires certain project settings. If you use the INtime wizards to develop the RT portion of your application, the wizards will set up your project properly. Settings vary, depending on whether you configured Microsoft Developer Studio to build a debug or a release version.

To view and verify settings required by INtime software, select Microsoft Developer Studio's Build>Settings menu option.

Only required RT portion settings are listed; checkboxes and radio buttons are disabled (not checked) unless otherwise noted. You can enter the values you want in fields for which no setting is listed.

General

Dbg	Rel	Field	Value
X	X	Microsoft Foundation Classes	Not using MFC

Debug

This tab requires no special settings.

Custom Build

This tab requires no special settings.

C/ EC++

Dbg	Rel	Category	Field	Value
X	X	Common options	Display only field; shows values derived from other fields on this tab.	
X		General		
X			Debug info	C7 compatible
	X			None
X			Optimizations	Disable
	X			Maximize Speed
X	X	Code Generation	Calling convention	__cdecl *
X	X	Precompiled headers	Not using precompiled headers	Enabled (checked)

X		Preprocessor	Preprocessor definitions	_WIN32 _DEBUG
	X			_WIN32
X	X		Additional include directories	c:\Program Files\INtime\rt\include
X	X		Ignore standard include paths	Enabled (checked)

Link

Dbg	Rel	Category	Field	Value
X	X	Project Options	Display only field; shows values derived from other fields on this tab. **Important**: You must add value: /heap:0x100000,0x2000	
X		General	Generate debug info	Enabled (checked)
X	X		Additions library path	c:\Program Files\INtime\rt\lib
X	X	Customize	Output filename	Must have an .RTA extension
X	X	Input	Object/library modules	ciff3m.lib rt.lib rtserv.lib pcibus.lib netiff3m.lib rmxiff3m.lib
		Output		
X	X	Stack allocations	Reserve Commit	0x100000 0x2000
X	X	Version information	Major Minor	21076 20052

Resources

This tab requires no special settings.

Browse Info

This tab requires no special settings.

Debug

You must debug both portions of your INtime application using the appropriate tools:

- **Windows portion**: Use standard Windows development tools, including the Microsoft Developer Studio debugger.
- **RT portion**: Use Soft-Scope plus the other debug tools provided with INtime software

Using the two debuggers, you can simultaneously view and debug on-target application code in both the Windows and the RT environments.

- **Soft-Scope (SSWIN32.EXE)**: A Windows-based RT debugger that supports on-target debugging of RT threads. The Soft-Scope debugger fully comprehends the RT constructs supported by INtime and supports dynamic debugging. Soft-Scope can debug multiple RT threads simultaneously while other threads continue to run.
- **System debug monitor (SDM)**: A command-line interface that provides low-level, static debugging capabilities. SDM requires a separate serial debug terminal for its user interface.

  For detailed information about using Soft-Scope, see Soft-Scope's Help. For detailed information about using SDM, select Debuggers>Low-level debugger>System Debug Monitor (SDM) in INtime Help.

- **INtex**: A Windows-based RT object browser. The INtex program, through its self-loaded RT counterpart, is able to show the state of all RT processes, threads, and objects. It can also display the code (via Microsoft Developer Studio) of a process that has been suspended due to a hardware fault (crash analysis).
- **INscope**: The INScope Real-time Performance Analyzer is a Windows application that allows you to trace execution of INtime applications. Trace information for thread switches, system calls, and interrupt handling is displayed in a graphical user interface with various tools available to allow system analysis.

Debugging tips

Performance monitor

To view CPU usage in both the Windows and RT kernels, you can run the Windows Performance monitor. Viewing CPU activity provides the feedback you need to determine if you divided labor appropriately between Windows

processes and RT processes. For more information about designing your INtime applications, see *Chapter 5, "Designing RT applications"*.

To view Windows and RT kernel activity in the Windows Performance Monitor:

1. Open the Performance Monitor by selecting this option from the Start menu:

   Control Panel>Administrative Tools>Performance>System Monitor

2. Open a chart by selecting the File menu's New Chart option.

3. Identify the performance metrics you want to view by choosing the Edit menu's Add to Chart option, then select these options:

Object	Counter	Purpose
Processor	% Processor time	Displays the percent of time devoted to Windows work.
INtime RT kernel	RT Kernel CPU usage (%)	Displays the percent of time devoted to RT work.

You can now run your INtime application and observe the CPU usage of both the Windows and RT kernels. When viewing the CPU usage, keep in mind that the Performance Monitor displays total CPU usage which includes more than the activity generated by an INtime application.

Status messages

When the RT kernel is running, a protection fault which occurs in the RT portion of an INtime application is handled by the default system hardware exception handler which suspends the faulting thread. Such threads display in the INtime Explorer with a Thread State of 2 and a Delay Request of 0xffff. For threads in this state, the CPU frame displays the thread's CPU context when the hardware fault occurred, including the CS:EIP address of the faulting instruction.

The error code indicates the fault encountered as follows:

Fault	Code	Description
EH_ZERO_DIVIDE	0x8100	Divide by Zero error
EH_SINGLE_STEP	0x8101	Single Step
EH_NMI	0x8102	NMI
EH_DEBUG_TRAP	0x8103	Debug Interrupt (Ignored by handler)
EH_OVERFLOW	0x8104	Overflow error
EH_ARRAY_BOUNDS	0x8105	Array Bounds error
EH_INVALID_OPCODE	0x8106	Invalid Opcode error
EH_DEVICE_NOT_PRESENT	0x8107	NPX device not present
EH_DOUBLE_FAULT	0x8108	Double Fault error

Fault	Code	Description
EH_DEVICE_ERROR	0x8109	NPX device error
EH_INVALID_TSS	0x810A	Invalid TSS error
EH_SEGMENT_NOT_PRESENT	0x810B	Segment Not Present error
EH_STACK_FAULT	0x810C	Stack Fault
EH_GENERAL_PROTECTION	0x810D	General Protection Fault
EH_PAGE_FAULT	0x810E	Page Fault

If Soft-Scope is running when the protection fault occurs, Soft-Scope traps the fault and allows you to debug or delete the offending program.

Prepare for release

Before you begin

- On the target system, ensure that you are logged on with Administrator privileges.
- Exit all programs prior to installing INtime software.
- If your system has a previously installed version of INtime software, remove it using the Add/Remove Programs Applet in the System Control Panel (select: Start>Control Panel>Add/Remove Programs. Highlight INtime 2.14 (or earlier) program, and then click Remove). Make sure none of the INtime services are running. If they are running, you must make sure they are set to Manual Start (using the Start>Control Panel>Administrative User/Services applet) and reboot the system before you can successfully complete the uninstall operation.

Using Runtime220.msi

To install INtime runtime software:

1. Start the runtime220.msi program:
   A. Select the Start>Settings>Control Panel>Add/Remove Programs Applet.
   B. Select runtime220.msi. You can use the Browse feature to quickly find the runtime220.msi program that originally came on the INtime 2.20 CD-ROM.

      If the Installation program detects a previous version of INtime software, it prompts you to exit the installation and uninstall the previous version.

      If the HAL currently installed on your machine is not compatible with INtime software, the Installation program prompts you to change the HAL driver.

For details, see *About the modified Windows HAL* in *Chapter 2, Understanding INtime software architecture*.

If your system does not have the Microsoft Installer product, the Installation program prompts you to install it. After installation, you may need to reboot.

The Installation program automatically resumes.

2. Review the INtime License and note its terms and conditions.
3. Select a destination directory for the INtime software files, then click the Next button. The default is c:\Program Files\INtime.
4. Select one of the installation options listed below, then follow the prompts in the installation..

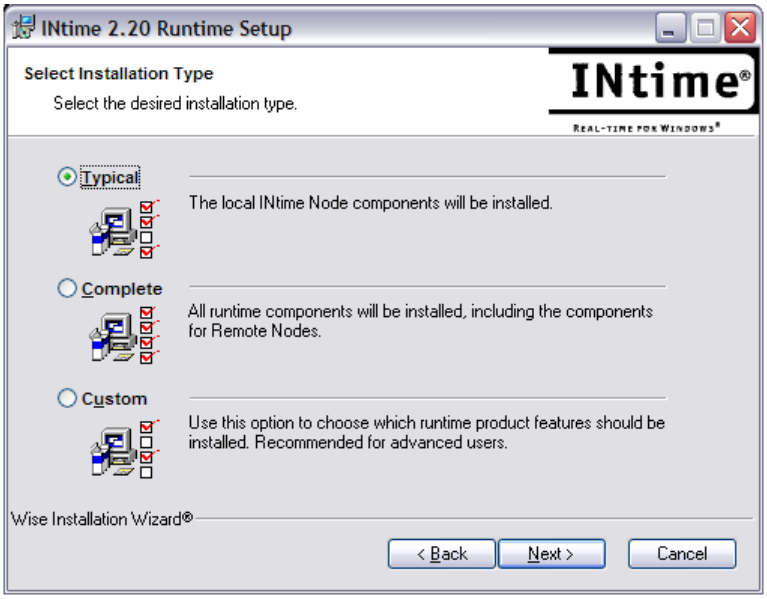

INtime 2.20 Runtime Setup
Select Installation Type
Select the desired installation type.
INtime
Typical
The local INtime Node components will be installed.
Complete
All runtime components will be installed, including the components for Remote Nodes.
Custom
Use this option to choose which runtime product features should be installed. Recommended for advanced users.
Wise Installation Wizard®
< Back
Next >
Cancel

Figure 10-2. Installing INtime Runtime software

Option	Description
Typical	Installs the INtime node runtime components. Choose this option to run INtime applications on this system.
Full	Installs all product runtime components. Choose this option to run INtime applications on this system, as well as configure and run INtime applications on one or more Remote INtime nodes.
Custom	Installs the runtime component groups you specify. Choose this option to make this system a Windows host that communicates with INtime applications on Remote INtime nodes.

5. Click the Next button to install the software.

   The runtime220.msi program creates the directory you specified, then installs the INtime software files.

6. Click the Finish button to complete the installation process.

7. Restart your system using one of these methods:

   - Select OK at the last screen. Runtime restarts your system.
   - Restart your system at a later time. You must restart your system before running INtime software.

The runtime220.msi executable installs those portions of INtime software that you can include in your derivative works. Use of runtime220.msi is governed by the INtime Software Redistribution License Agreement you must enter into in order to include INtime Software in your product. For information about obtaining a Software Redistribution License and on the associated per unit royalty due to TenAsys Corporation for use of the INtime software in your product, contact support@tenasys.com.

Sample INtime applications

INtime software has a number of sample applications that you can use as samples for your own INtime applications.

The source code for these applications are provided in MSVC 6.0 generation format, and reside in separate directories per demo. For instance, the INtime API test program source files reside in the c:\Program Files\INtime\Projects\rttest directory.

You can open the projects for these applications from the Start>All Programs>INtime>Sample Code and Projects menu. The executables generated by these projects can be loaded using the INtime RT Application Loader (Start>All Programs>INtime>RT Application Loader).

EventMsg DLL Project

This DLL allows you to customize event messages.

Item	Source
Pathname	c:\Program Files\INtime\Projects\eventmsg
Runtime requirements	The EventMsg Project builds the EventMsg.DLL file properly only if you first build the project under release mode from within the Microsoft Developer Studio. After building the project under release mode, you may then build the project under debug mode.

INtime API Sample

This test application exercises most INtime software system calls.

Item	Source	Executable
Pathname	c:\Program Files\INtime\projects\rttest	c:\Program Files\INtime\projects\rttest\debug

INtime Serial Driver Sample

This sample application consists of a working INtime serial driver plus an INtime application that utilizes the driver to communicate with a serial terminal connected to the specified COMn channel. The driver provides an RT application with high-speed serial I/O communications capabilities.

The driver (c:\Program Files\INtime\Projects\serialio\debug\serdrvr.rta) operates as an INtime service that manages a serial device attached to COM1 or COM2 on an INtime system. The demo (c:\Program Files\INtime\Projects\serialio\debug\serialio.rta) uses an INtime Message Port to obtain read/write access to the COMn channel managed by the Serdrvr.rta service. Both components are provided in source and binary form. For more information, see the README.TXT file in the source code directory for this sample application.

Item	Source	Executable
Pathname	c:\Program Files\INtime\projects\serialio	c:\Program Files\INtime\projects\serialio\debug

INtime Graphical Jitter

This application measures the minimum, maximum, and average times between low-level ticks via an Alarm Event Handler. Because this application is made from both an RT and Windows executables, it shows both INtime and NTX API usage.

INtime Graphical Jitter includes these executables:

- **Jitter.exe**: The Windows executable. Automatically starts ClkJitr0Rt.rta, then processes output and displays a histogram.
- **ClkJitr0Rt.rta**: The RT executable. Started by Jitter.exe.

Item	Source	Executable
Pathname	c:\Program Files\INtime\projects\jitternt	c:\Program Files\INtime\projects\jitternt\debug
	c:\Program Files\INtime\projects\jitterrt	c:\Program Files\INtime\projects\jitterrt\debug
Invocation	Invoke the Windows jitter.exe program.	

Real-time Interrupt Sample

This application tests the INtime RT Interrupt system calls using the Transmitter Ready interrupt from COM1.

The INtime RT Interrupt API Test takes over COM1 and toggles its Transmitter Ready Interrupt. When the test ends, COM1 is disabled. Make sure COM1 is available on your system before running this test application. When you run the test, continuous activity occurs on the RT side, preempting Windows activity for eight 10-second time periods.

Item	Source	Executable
Pathname	c:\Program Files\INtime\ projects\inttest	c:\Program Files\INtime\projects\ inttest\debug

C and C++ Samples for Debugger

These simple C and C++ programs are provided as a vehicle to demonstrate the Soft-Scope debugger's capabilities. The C++ program also demonstrates several components of the C++ language available to RT applications, as well as basic classes, dynamic instantiation, operator overloading, and so on. It also shows the libraries and startup modules needed.

Item	Source	Executable
Pathname	c:\Program Files\INtime\ projects\csamp (C program) c:\Program Files\INtime\ projects\cppsamp (C++ program)	c:\Program Files\INtime\projects\ csamp\debug c:\Program Files\INtime\projects\ cppsamp\debug
Invocation	Use Soft-Scope to load these programs.	

TCPIP>TCP-IP Server Sample

Sample TCP/IP application that waits for 130Kbyte messages and returns them to the sender.

Item	Source	Executable
Pathname	c:\Program Files\INtime\ projects\tcpserv	c:\Program Files\INtime\projects\ tcpserv\debug
Invocation	See the ReadMe.txt file in the source code directory.	

TCPIP>TCIP-IP Client Sample

Sample TCP/IP application that looks for a TCP/IP server and then sends 130KByte messages to it and waits for the reply.

Item	Source	Executable
Pathname	c:\Program Files\INtime\projects\tcpclnt	c:\Program Files\INtime\projects\tcpInt\debug
Invocation	See the ReadMe.txt file in the source code directory.	

Fault Handling (ntrobust)

This INtime application has both a Windows and a RT portion. The Windows portion allows the user to set up timing parameters that control how often a thread in the RT portion causes a hardware fault. The application demonstrates how another RT thread can detect and log the failure, delete the offending thread, and recreate it, all without affecting Windows or other RT processes.

Item	Source	Executable
Pathname	c:\Program Files\INtime\projects\NtRobust	c:\Program Files\INtime\projects\NtRobust\Debug
	c:\Program Files\INtime\projects\NtRobust\RtRobust\	c:\Program Files\INtime\projects\NtRobust\RtRobust\Debug
Invocation	See the ReadMe.txt file in the source code directory.	

Floating Point Exception Handling

This simple program demonstrates floating point exception handling.

Item	Source	Executable
Pathname	c:\Program Files\INtime\projects\FpExcep	c:\Program Files\INtime\projects\FpExcep\Debug
Invocation	See the ReadMe.txt file in the source code directory.	

RSL Example

This RT program demonstrates the creation and use of RT Shared Libraries, the RT analog for Windows DLLs.

Item	Source	Executable
Pathname	c:\Program Files\INtime\projects\RslTest	c:\Program Files\INtime\projects\RslTest\Debug
Invocation	See the ReadMe.txt file in the source code directory.	

NTX Sample (MsgBoxDemo)

This INtime application has both a Windows and a RT portion. The Windows portion looks up an RT mailbox created by the RT portion, and then waits at the mailbox. When an RT thread sends a message to the mailbox, the Windows portion displays the received data in a message box on the Windows side. RT semaphore and RT shared memory usage are also demonstrated.

Item	Source	Executable
Pathname	c:\Program Files\INtime\projects\MsgBoxDemo c:\Program Files\INtime\projects\MsgBoxDemo\RtMsgBox	c:\Program Files\INtime\projects\MsgBoxDemo\Debug C:\Program Files\INtime\projects\MsgBoxDemo\RtMsgBox\Debug
Invocation	See the ReadMe.txt file in the source code directory.	

INtime Windows STOP Detection sample (STOPmgr)

This sample application shows how an INtime application can detect either a Windows Crash (blue screen) or Windows Shutdown event and prevent Windows from completing its normal actions until the RT application has had a chance to do a “graceful” shutdown.

Item	Source	Executable
Pathname	c:\Program Files\INtime\projects\stopmgr	c:\Program Files\INtime\projects\stopmgr\debug

INtime USB Client sample

This sample application demonstrates how to use the INtime USB subsystem. It monitors a USB keyboard and prints a dump of each keystroke as it occurs.

Item	Source	Executable
Pathname	c:\Program Files\INtime\projects\usbsamp	c:\Program Files\INtime\projects\usbsamp\debug

III Appendices

The appendices include:

Appendix A: INtime software system calls

Lists and describes system calls that threads in the RT portion of INtime applications use to communicate with each other and with Windows threads. You can find detailed information, including syntax and parameter values, in Help.

Appendix B: INtime software components

Lists and describes INtime software program files.

Appendix C: Adding INtime software to an XP Embedded configuration

Lists and describes how to add INtime components to a Windows XP Embedded development environment so that XP Embedded images can be produced that include these INtime components.

Appendix D: Troubleshooting

Lists problems you may encounter while running INtime software, and explains how to avoid or resolve those problems.

A INtime software system calls

This appendix lists and describes INtime software system calls. Use this appendix to identify the system calls you want to use for each RT kernel exchange object. The calls are arranged first by system call types, and then by objects available to that object type: NTX, high-, or low-level. Other calls are listed at the end of the appendix.

For detailed information about system calls, including syntax and parameter values, refer to Help. For more information about accessing Help, see *Where to get more information* on page page v

Object	Page
System call types	108
RT system calls	112
Exception handling	112
Interrupts	112
Mailboxes	113
Memory	114
Object directories	116
Ports	117
Processes	119
Regions	120
Scheduler	120
Semaphores	121
Status	122
System accounting	123
System data	123
TCP/IP calls	124
Threads	125
Time management	126
RT services	127
RT service handlers	129
RT service calls	128
PCI library calls	130

System call types

Several types of calls exist, and each kernel exchange object has calls of one or more type associated with it:

- **Windows eXtension (NTX) calls**: Windows uses these calls to communicate with the INtime kernel. NTX calls allow Windows applications to operate on objects created by, stored in, and controlled by the RT kernel.
- **Real-time (RT) calls**: The RT kernel uses these calls to run the RT portion of INtime applications and communicate with Windows. INtime software provides two levels of system calls for RT kernel exchange objects:
  - **High-level (validating) calls**: Write, test, and debug your application using high-level calls with their protection and validation features.
  - **Low-level (non-validating) calls**: When the application runs as you desire, increase performance by substituting low-level system calls where appropriate.
- **RT services**: INtime real-time applications (RTAs) which process messages from a user and events from an interface, and the service handlers required to perform service-dependent functions of the system calls.

RT kernel objects provide these RT call levels:

Object	NTX	High-level	Low-level
Distributed system manager	X	X	
Exception handling		X	
Interrupts		X	
Mailboxes	X	X	X
Memory	X	X	
Object directories	X	X	
Ports	X	X	
Processes	X	X	
Regions		X	
Scheduler			X
Semaphores	X	X	X
Status	X	X	
System accounting		X	
Threads		X	
Time management			X

Regarding INtime software system calls:

- System call names correspond to related Windows functions, where appropriate. Where it is confusing to associate the function of an INtime software object with a Windows object, the names reflect the difference.

  For example, no Windows object corresponds closely to an INtime software region, so these are left as regions. On the other hand, INtime software alarms are encapsulated as an object which somewhat correspond to a specialized Windows event object, so this is called AlarmEvent. It does not, however, perform all the same functions as a Windows event.

- To avoid confusion with similarly-named Windows calls, INtime software system call names usually include the letters 'Rt'.

- RT kernel objects and the kernel interface which differ substantially from corresponding items in the other INtime software layers, include a 'kn' prefix to indicate calls that map to low-level RT kernel functions and deal with low-level kernel objects.

NTX calls

NTX calls allow Windows applications to operate on objects created by, stored in, and controlled by the RT kernel. This allows Windows and INtime applications to communicate and coordinate their activities.

Handle conversion

The NTX DLL converts in transit NTXHANDLES sent to the RT kernel to RTHANDLES, and vice-versa. This is necessary as the RT kernel cannot operate on NTXHANDLES and NTX can't operate on RTHANDLES.

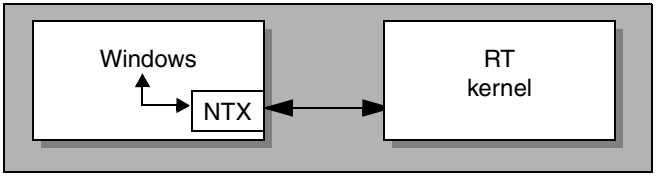


Figure A-1. Converting NTXHANDLES to RTHANDLES

If a handle passes from RT to Windows in data, ntxImportRtHandle must convert it. Object directories and object mailboxes are converted automatically.

RT calls

You use these calls when developing the real-time portion of an INtime application.

High-level (validating) calls

High-level calls provide lower performance, plus higher protection and validation features. Memory is allocated automatically from the process's pool. Each high level object consumes a slot from the system GDT (8000 objects maximum)

High-level system calls validate a call's parameters; a condition code returned by the call indicates whether you used invalid parameters. Condition codes for trying to read or write memory to which you have no access also exist.

High-level exchange objects validate parameters and are protected against unexpected deletion. High-level calls exist for these exchange objects:

- Object and data mailboxes
- Counting semaphores
- Regions (for mutual exclusion with priority inversion protection)

Low-level (non-validating) calls

Low-level calls provide higher performance, but lower protection and validation features. Low-level objects provide functionality beyond that of high-level objects.

You must allocate memory for low-level objects and may allocate memory beyond low-level object needs. You can use this additional memory to store application-specific state information associated with the object.

Low-level objects are not protected against unexpected deletion and do not validate parameters (if you need parameter validation, use high-level system calls). Low-level calls use the flat, 4 Gbyte addressing capabilities of the microprocessor. They do not use segmentation. Therefore, they do not consume a slot from the system GDT.

Use low-level objects in these situations:

- For well-tested code.
- For isolated parts of the application, such as signaling ordinary threads from an interrupt handler.

- When performance is critical, such as high-performance, unvalidated sending and receiving of data mailbox messages and semaphore units..

System calls that manipulate low-level objects assume that all memory reference pointers received are valid.

Low-level calls exist for these exchange objects:

- Data mailboxes
- Single-unit semaphores
- Region semaphores (with priority inversion protection)
- Software alarm events (virtual timers) that invoke alarm event threads that you write.

RT services

Real-time services include:

- **RT service calls**: An INtime real-time application (RTA) which processes messages from a user and events from an interface. A service is defined by the set of messages and the actions provoked by those messages. Each interface is associated with a service descriptor.
- **RT service handlers**: Subroutines invoked by the kernel to perform service-dependent functions of the system calls.

RT system calls

Exception handling

High-level calls

System call	Description
SetRtExceptionHandler	Assigns an exception handler and exception mode or changes the current mode for any of the following: • Current thread exception handler • Current process exception handler • System-wide exception handler
GetRtExceptionHandlerInfo	The GetRtExceptionHandlerInfo function returns the address and exception-handling mode for any of the following: • Current thread's exception handler • Current process' exception handler • System-wide exception handler • System-wide hardware exception handler (trap handler)

Interrupts

This group provides the system calls required to manage interrupts. Interrupt requests are encoded to indicate their source; the resulting coding is called the interrupt level. This encoded interrupt level is required in a number of the system calls in this group. Macros are provided in the header files for the 15 standard PC interrupt levels, IRQ0_LEVEL to IRQ15_LEVEL.

High-level calls

System call	Description
SetRtInterruptHandler SetRtInterruptHandlerEx	Assigns an interrupt handler to the specified interrupt level, and optionally makes the calling thread the interrupt thread for that level.
ResetRtInterruptHandler	Cancels the assignment of the current interrupt handler to the specified level and disables the level.
EnterRtInterrupt	Allows the interrupt handler to have access to static data belonging to the RT process which owns the handler. This function is called from an interrupt handler.
GetRtInterruptLevel	Returns to the calling thread the highest (numerically lowest) level that an interrupt handler has started servicing but has not yet finished.

SignalRtInterruptThread	Sends an EOI signal to the interrupt hardware then schedules the interrupt thread associated with the specified level. This function is called from an interrupt handler.
SignalEndOfRtInterrupt	Sends an EOI signal to the interrupt hardware. This function is called from an interrupt handler.
WaitForRtInterrupt	Used by an interrupt thread to signal its readiness to service an interrupt. It blocks for the given number of milliseconds.
DisableRtInterrupt	Disables the specified interrupt level. It has no effect on other levels.
EnableRtInterrupt	Enables a specific interrupt level which must have an interrupt handler assigned to it.

Mailboxes

These calls manage various RT kernel mailbox object types.

INtime software includes two kinds of RT mailboxes:

- Object mailboxes manage RTHANDLES.
- Data mailboxes manage short sections of arbitrary data.

NTX calls

System call	Description
ntxCreateRtMailbox	Creates an RT mailbox.
ntxDeleteRtMailbox	Deletes an RT mailbox.
ntxReceiveRtHandle	Receives RTHANDLES from an object mailbox.
ntxSendRtHandle	Sends RTHANDLES to an object mailbox.
ntxSendRtData	Copies arbitrary data to a data mailbox (up to 128 bytes).
ntxReceiveRtData	Copies arbitrary data out of a data mailbox (up to 128 bytes).

High-level calls

System call	Description
CreateRtMailbox	Creates a new mailbox and returns an RTHANDLE for the object. The mailbox type is determined by the flags parameter.
DeleteRtMailbox	Deletes the mailbox specified by the RTHANDLE given.
SendRtHandle	Sends an RT object RTHANDLE to a mailbox which has been created to pass RT objects.

ReceiveRtHandle	Receives an RTHANDLE from an object mailbox.
SendRtData	Copies arbitrary data to a data mailbox (up to 128 bytes).
ReceiveRtData	Copies arbitrary data out of a data mailbox (up to 128 bytes).

Low-level calls

System call	Description
knCreateRtMailbox	Creates a kernel mailbox with the given specifications.
knDeleteRtMailbox	Deletes the kernel mailbox associated with the given kernel handle.
knSendRtData	Sends a data message to the given kernel mailbox.
knSendRtPriorityData	Sends high-priority data to a kernel mailbox, bypassing the queue.
knWaitForRtData	Requests a message from a specific kernel mailbox.

Memory

The NTX and RT portions of INtime applications can share regions of memory created by the RT portion of INtime applications.

These system calls implement the flat-model memory management interface for RT applications. There are additional calls for the management of page-aligned segments for sharing between applications (both RT and Windows applications). Also included are the calls that an application requires to allocate memory from its own virtual segment.

Physical memory may be allocated and mapped into an application's virtual segment and then an RTHANDLE created for this allocated memory so that it may be shared between applications (both RT and Windows applications)

NTX calls

System call	Description
ntxMapRtSharedMemory ntxMapRtSharedMemoryEx	Obtains a Windows pointer to the section of RT memory defined by the call's SharedRTMemoryHandle parameter.
ntxGetRtSize	Determines the size of the call's RT object parameter
ntxUnmapRtSharedMemory	Removes the mapping of a RT memory section and returns Windows system resources mapped to that memory.
ntxCopyRtData	Copies data directly to/from Windows application space from/to an RT memory object.

High-level calls

System call	Description
AllocateRtMemory	Allocates memory from the current process's memory pool to the current thread's virtual segment.
FreeRtMemory	Frees physical memory associated with the calling thread's virtual segment.
CreateRtMemoryHandle	Creates a handle for an area of memory in the process's virtual segment.
DeleteRtMemoryHandle	Deletes a memory handle created with CreateRtMemoryHandle.
MapRtSharedMemory	Maps a memory area, previously created by another process using CreateRtMemoryHandle, into the current thread's memory space.
MapRtPhysicalMemory	Maps a physical memory area, defined by its absolute address and contiguous length before any translation imposed by the paging system, into the current process' virtual segment.
GetRtPhysicalAddress	Returns the physical address for a valid buffer described by the call parameters.
GetRtSize	Returns the number of bytes in a previously allocated segment.
CreateRtHeap	Creates a heap object by allocating a segment of *cbHeapSize* bytes (plus overhead) and creating the heap structure in this segment.
DeleteRtHeap	Deletes a heap object, given its handle. Any flat-model mappings are deleted.

RequestRtBuffer	Allocates a memory buffer from a heap object and returns a pointer to the caller.
ReleaseRtBuffer	Returns a previously-allocated buffer to its heap.
GetRtHeapInfo	Returns a structure containing information about a heap object.
GetRtBufferSize	Returns the allocated size of a buffer previously allocated from a heap.

Object directories

The object directory provides a rendezvous mechanism between RT threads or between an RT thread and a Windows thread in an INtime application. The RT portion of an INtime application creates shared objects and catalogs those it wants to share. The Windows portion of the application awaits their cataloging.

These system calls manage RT object directories. Objects may be cataloged, looked up, and uncataloged. The default directory is the one for the current process. Handles for the other processes may be obtained using the GetRtThreadHandles call. Catalog names may be up to 14 characters long and are case-sensitive.

NTX calls

System call	Description
ntxLookupNtxhandle	Looks up a name-to-handle association. Given a name, an RT or NTX application can look up the handle associated with it.
ntxImportRthandle	Obtains an NTXHANDLE corresponding to an RTHANDLE.
ntxCatalogRtHandle	Creates a name-to-handle association in the object directory of the RT process specified in the call.
ntxUncatalogRtHandle	Removes the name-to-handle association in the object directory of the RT process specified in the call.
ntxGetRootRtProcess	Obtains the root RT process handle for the NTX location specified in the call.
ntxGetType	Checks the RT object's type.

High-level calls

System call	Description
CatalogRtHandle	Creates a name-to-handle association in the object directory of the RT process specified in the call.
LookupRtHandle	Searches the given process' object directory for a given name and returns the object handle, if found.
UncatalogRtHandle	Removes an entry from a process' object directory.
InspectRtProcessDirectory	Returns the contents of a process' object directory.
GetRtHandleType GetRtHandleTypeEx	Returns a value indicating the type of an RT object. The handle must be for a valid RT object.

Ports

NTX calls

System call	Description
ntxBindRtPort	Binds an address to a port.
ntxCreateRtPort	Creates a port for access to an RT service module.
ntxDeleteRtPort	Destroys an RT port created by a Windows process.
ntxConnectRtPort	Creates a connection between one local port identified by a handle, and another port identified by an address.
ntxAttachRtPort	Forwards messages from one port to another on the same service.
ntxDetachRtPort	Stops forwarding messages from the specified port.
ntxGetRtPortAttributes	Obtains information about the specified port.
ntxSendRtMessage	Sends a message from a port to a service.
ntxSendRtMessageRSVP	Sends the request part of an RSVP transaction.
ntxCancelRtTransaction	Cancels an RSVP message transmission in progress, or the status reporting phase of an asynchronous send operation.
ntxReceiveRtMessage	Receives a message at a port.
ntxReceiveRtReply	Receives the reply phase of an RSVP transaction.
ntxGetRtServiceAttributes	Allows the caller to receive parameters from the service.

ntxSetRtServiceAttributes	Allows the caller to specify run-time parameters for the service.
ntxRequestRtBuffer	Allocates a memory buffer from the heap object associated with a port connected to a service, and returns a pointer to the Windows mapping of the buffer.
ntxReleaseRtBuffer	Returns memory to the heap object from which it was taken.

Service support

System call	Description
InstallRtServiceDescriptor	Adds a service to the operating system by linking the service descriptor to the service descriptor list.
UninstallRtServiceDescriptor	Removes a service from the operating system by unlinking the service descriptor from the service descriptor list.
GetRtServiceAttributes	Interrogates certain attributes of the interface controlled by the service.
SetRtServiceAttributes	Sets or changes some attributes of the interface controlled by the service.

Port object management

System call	Description
AttachRtHeap	Makes a heap's memory resources available to one or more message port objects.
AttachRtPort	Enables an application to monitor several ports simultaneously.
BindRtPort	Binds an address to a port.
ConnectRtPort	Creates a connection between a local port and a remote port.
CreateRtPort CreateRtPortEx	Creates a message port for access to a given service.
DeleteRtPort	Deletes a port object. Any messages queued at the port are discarded and, if the port is forwarded, forwarding is severed.
DetachRtHeap	Ends the association between a heap object and a message port.
DetachRtPort	Ends message forwarding from the specified message port.
GetRtPortAttributes	Returns a structure giving information about the port object indicated by the supplied handle.

Message transmission

System call	Description
SendRtMessage	Sends a data message from a port to a service.
SendRtMessageRSVP	Sends the request phase of a transaction and allocates a storage for the response part of the transaction.
SendRtReply	Sends a response message to an earlier receive SendRtMessageRSVP message.
CancelRtTransaction	Performs synchronous cancellation of RSVP message transmission.
ReceiveRtMessage	Receives a message at a port.
ReceiveRtReply	Receives a reply message to an earlier RSVP transmission. The port cannot be a sink port.
ReceiveRtFragment	Receives a fragment of an RSVP data message request.

Processes

NTX calls

System call	Description
ntxCreateRtProcess	Creates a process.
ntxRegisterDependency	Creates a dependency relationship between the calling process and the specified sponsor.
ntxUnregisterDependency	Removes the dependency relationship between the calling process and the specified sponsor.
ntxRegisterSponsor	Registers the calling process as a Sponsor with the given name.
ntxUnregisterSponsor	Removes the current sponsor name from the active sponsor state.
ntxNotifyEvent	Blocks until one of the desired notifications has been received.

High-level calls

System call	Description
ExitRtProcess	Deletes the current process, all of the process' threads, and all objects created by the threads.
RegisterRtDependency	Looks up the name in the sponsor list and creates a dependency relationship to that sponsor process.
UnregisterRtDependency	Removes the dependency relationship from the database between the RT process and the Windows sponsor registered with the given name.
RegisterRtSponsor	Allows the RT process to register as a sponsor under the given name.
UnregisterRtSponsor	Removes the RT process registered as a sponsor from the database

Regions

High-level calls

System call	Description
CreateRtRegion	Creates a region object.
DeleteRtRegion	Deletes a region object.
AcceptRtControl	Receives ownership of a region object only if it is immediately available.
WaitForRtControl	Gains ownership of a region. This function blocks until the current owner gives up the region.
ReleaseRtControl	Releases this thread’s most recently obtained region object.

Scheduler

Low-level calls

System call	Description
knRtSleep	Puts the calling thread to sleep for the specified number of kernel ticks.
knStartRtScheduler	Cancels one scheduling lock imposed by the knStopRtScheduler function.
knStopRtScheduler	Temporarily locks the scheduling mechanism or places an additional lock on the mechanism for the running thread.

Semaphores

Semaphores contain units. These system calls deal with semaphore objects and the units associated with them.

RT semaphores differ from Windows semaphore objects in these respects:

- Choice of FIFO or priority queuing.
- A thread may wait for multiple units from a semaphore.
- Waiting times are defined as “wait for the specified number of clock ticks” rather than “wait for at least the specified number of milliseconds”.
- Multiple-object waiting is not supported. High-level semaphores differ from the low-level semaphores in the following respects:
  - High-level semaphores have parameter validation..
  - Multiple units may be sent to and received from high-level semaphores.
  - High-level semaphore may be created with any initial count other than one or zero (low-level semaphores may be created only with zero or one unit).
  - Low-level handles may not be cataloged.

NTX calls

System call	Description
ntxCreateRtSemaphore	Creates an RT semaphore.
ntxDeleteRtSemaphore	Deletes an RT semaphore.
ntxWaitForRtSemaphore	Waits for and removes units from an RT semaphore.
ntxReleaseRtSemaphore	Adds units to an RT semaphore.

High-level calls

System call	Description
CreateRtSemaphore	Creates a semaphore with the given initial and maximum number of units.
DeleteRtSemaphore	Deletes a semaphore.
WaitForRtSemaphore	Waits for and removes a specified number of units from a semaphore.
ReleaseRtSemaphore	Sends a given number of units to a semaphore.

Low-level calls

System call	Description
knCreateRtSemaphore	Creates 1 of 3 kinds of kernel semaphores with 0 or 1 initial units.
knDeleteRtSemaphore	Deletes a kernel semaphore.
knWaitForRtSemaphore	Waits for and removes a unit from the specified kernel semaphore.
knReleaseRtSemaphore	Sends a single unit to a specified kernel semaphore.

Status

NTX calls

System call	Description
ntxGetLastRtError	Returns the status code of the last failing NTX call made in this Windows thread.
ntxGetRtErrorName	Returns a pointer to a constant string.
ntxLoadRtErrorString	Copies a short sentence (no punctuation) that describes "Status" into the buffer at *lpBuffer*.
ntxGetRtStatus	Verifies whether the RT kernel is currently running.

High-level calls

System call	Description
GetLastRtError	Returns the status code of the last failing RT system call made by this RT thread.
SetLastRtError	Sets the calling thread's last error field.
CopyRtSystemInfo	Copies the contents of the RQSYSINFO segment, cataloged in the root process, into the SYSINFO structure
ReportRtEvent	Collects log data in the same format as the Windows ReportEvent function, and passes it to the INtime Registry Service for logging in the Windows Registry.

System accounting

High-level calls

System call	Description
GetRtThreadAccounting	Returns information about when a thread was created and the amount of time the thread has run.
SetRtSystemAccountingMode	Toggles accounting tracking. You can return accounting information using GetRtThreadAccounting.

System data

NTX calls

System call	Description
ntxGetLocationByName	Gets a handle to a specified location.
ntxGetFirstLocation	Returns a handle to the first known location.
ntxGetNextLocation	Gets the handle that follows the one return by the last location call.
ntxGetNameOfLocation	Gets the name NTX uses for a handle.
NtxFindINtimeNode	Invokes the INtime RT Client Browser to allow target INtime node selection.

TCP/IP calls

System call	Description
accept	Accepts a connection on a socket.
bind	Assigns a name to an unnamed socket.
bstring	Executes binary string operations.
byteorder	Converts short and long quantities between network byte order and host byte order.
connect	Initiates a connection on a socket.
gethostname	Gets and sets the local host name.
getpeername	Returns the socket name of the connected remote socket.
getsockname	Returns the current name for the specified socket.
getsockopt	Returns or sets options associated with a socket.
inet	Manipulates Internet addresses.
listen	Listens for connection requests on a socket.
recv	Receives a message from a socket.
select	Checks the sockets specified in the sets of descriptors supplied as parameters to see if any of the sockets are ready for receiving or sending, or have out-of-band data pending.
send	Sends a message from one socket to another.
shutdown	Shuts down all or part of a full-duplex connection.
socket	Creates an endpoint for communication.
socktout	Defines a maximum time to wait for completion of any subsequent calls on the specified socket.

Threads

High-level calls

System call	Description
CreateRtThread	Creates a thread to execute within the context of the calling process.
DeleteRtThread	Deletes a thread referenced by the given handle.
GetRtThreadPriority	Returns the specified thread's current priority.
GetRtThreadHandles	Returns a handle for either the calling thread, the calling thread's process, the parameter object of the calling thread's process, the root process, or the parent process of the calling thread's process, depending on the encoded request.
SetRtThreadPriority	Dynamically changes the priority of a non-interrupt thread. The new value must not exceed the containing process' maximum priority.
SetRtProcessMaxPriority	Dynamically change the maximum priority of threads in a process.
RtSleep	Places the current thread in the sleep state until the required number of system ticks have occurred. System ticks are always 10ms apart.
GetRtThreadAccounting	Returns information about when a thread was created and the amount of time the thread has run.
SuspendRtThread	Increases by one the suspension depth of a specified thread.
ResumeRtThread	Decreases by one the suspension depth of the specified non-interrupt thread.
GetRtThreadInfo	Returns information about a thread, including such items as priority, exception handler, containing process, and execution state.
GetRtThreadState	Returns information about the state of any thread in the system, including such items as the execution state and the CPU registers for that thread's execution context (if the thread has been suspended due to its causing a hardware fault).

Time management

INtime software provides low-level time management calls that allow threads to create alarm events and to sleep for a specified amount of time. The kernel also provides a RT clock.

An alarm event is an object which is signaled when a pre-determined time interval has expired. The alarm event mode may be single-shot or repeatable.

The kernel's RT clock is a counter that the kernel uses to keep track of the number of kernel clock ticks that have occurred. When the kernel is initialized, the count is set to 0 (zero). The period of a kernel clock tick is configurable via the INtime Configuration utility and you can read the current configuration using CopyRtSystemInfo.

Low-level calls

System call	Description
knCreateRtAlarmEvent	Creates an alarm event object which is triggered by an alarm.
knWaitForRtAlarmEvent	Waits at an alarm object for the given time interval, or until the alarm triggers.
knResetRtAlarmEvent	Resets a one-shot alarm after it has triggered.
knDeleteRtAlarmEvent	Deletes an alarm event object and releases the memory used to store its state for reuse.
knGetKernelTime	Returns the value of the counter the kernel uses to tally the number of low-level ticks that have occurred.
knSetKernelTime	Sets the value of the counter that the kernel uses to tally the number of low-level ticks that have occurred.

RT services

Registry calls

System call	Description
RtRegCloseKey	Releases a handle to a key.
RtRegConnectRegistry	Establishes a connection to a registry handle on another computer.
RtRegCreateKeyEx	Creates a key.
RtRegDeleteKey	Deletes a subkey from the registry.
RtRegDeleteValue	Deletes a value from a registry key.
RtRegEnumKeyEx	Enumerates subkeys of an open registry key.
RtRegEnumValue	Enumerates a value for an open registry key.
RtRegFlushKey	Writes attributes of an open key into the registry.
RtRegLoadKey	Creates a subkey and stores registration information into that subkey.
RtRegOpenKeyEx	Opens a key.
RtRegQueryInfoKey	Retrieves information about a registry key.
RtRegQueryValueEx	Retrieves type and data for a value name associated with an open registry key.
RtRegReplaceKey	Replaces the file backing a key and all its subkeys with another file.
RtRegRestoreKey	Reads registry information in a file and copy it over a key.
RtRegSaveKey	Saves a key and all its subkeys and values to a new file.
RtRegSetValueEx	Sets the data and type of a value under a registry key.
RtRegUnLoadKey	Unloads a key and subkeys from the registry.

RT service calls

System call	Description
RequestControlBuffer	Requests a control buffer from the service pool.
ReleaseControlBuffer	Returns a control buffer to the service pool.
RequestTransaction	Requests a TRANSACTION buffer from the service transaction pool.
ReleaseTransaction	Returns a transaction structure to the pool.
LookupPortHandle	Looks up a port handle given a port ID.
GetTransaction	Upon receiving a response message from the interface, instructs the service thread to tie the message with its transaction structure.
GetRSVPTransaction	Upon receiving a response message from the interface, instructs the service thread to tie the message with its transaction structure.
DeliverMessage	Delivers a complete transactionless message to a port. This call is typically made from the service thread.
DeliverTransaction	Delivers a transaction. This call is used after a service thread receives a response message, and the transaction is complete.
DeliverStatus	Terminates a transmit operation, usually from the service thread.
EnqueueOutPutTransaction	Enqueues a transaction on the service output queue when (for example) the transmitter hardware is busy.
QueryOutputTransactionQueue	Returns the transaction at the head of the service output queue.
DequeueOutputTransaction	Dequeues the transaction at the head of the service output queue and returns a pointer to it.
EnqueueInputTransaction	Enqueues a transaction on the service input queue. This allows the service to maintain an ordered list of requests for later completion by the service thread.
QueryInputTransactionQueue	Returns the transaction at the head of the service input queue.
DequeueInputTransaction	Dequeues the transaction at the head of the service input queue and returns a pointer to it.
GetPortId	Returns the port ID for a given port handle.
SetPortParameter	Sets the port parameter for the given port to a value given by the caller.
GetPortParameter	Retrieves the parameter previously associated with a port by a call to SetPortParameter.

EnterServiceRegion	Enters the region associated with the service. Currently the SendMessage handler is called while in this region. If mutual exclusion is desired between the service thread and the SendMessage handler, the service thread can make this call.
ExitServiceRegion	Exits the service region previously entered with EnterServiceRegion.

RT service handlers

Service handlers are subroutines invoked by the kernel to perform service-dependent functions of the system calls.

For example, calling SendRtMessage causes the kernel to invoke the handler supplied by the service to handle the transmission of the message to the interface. Some of these handlers must be supplied by the service while others are optional.

The control and transaction buffer pools are the static resources used for allocating internal data structures. The size of these pools is determined from parameters in the service descriptor at installation time.

System call	Description
CancelTransaction	implement special actions as a result of calling CancelRtTransaction or DeleteRtPort.
CreatePort	Invoked when CreateRtPort is called. A status of E_OK must be returned for the port to be created, else the port object is deleted and the status code is returned to the caller.
DeletePort	Invoked when DeleteRtPort is called. It must return a status of E_OK for the port to be deleted, else the port object is deleted and the status code is returned to the caller.
Finish	Ensures that any service-dependent resources are cleaned up before the service process is killed.
GetAttributes	Gets attributes from the service when an application calls GetRtServiceAttributes.
GetFragment	Invoked when an application calls ReceiveRtFragment.
Initialize	Performs any service-specific initialization functions and returns a suitable status code to the caller.

SendMessage	Implemented by all services which require a transmission function.
Service	Invoked by the service thread when an event occurs.
SetAttributes	Passes parameters from SetRtServiceAttributes.
UpdateReceiveInfo	Invoked just before returning results to either ReceiveRtMessage or ReceiveRtReply, caused by a service handler calling DeliverStatus. The kernel handles the receipt of a message at a port, and then may call this routine in order that the RECEIVEINFO structure may be filled out.
VerifyAddress	Validates the address parameter passed to the call.

PCI library calls

System call	Description
PciInitialize	Initializes the PCI library by determining the PCI configuration access method used by the local chipset.
PciReadHeader	Reads the PCI configuration header fields to the supplied PCI_DEV structure.
PciFindDevice	Locates a PCI device given the vendor and device IDs, and an instance number.
PciSetConfigRegister	Writes a value to a given PCI configuration register.
PciGetConfigRegister	Reads a value from a given PCI configuration register.
PciVendorName	Returns a text string corresponding to the vendor ID supplied as a parameter.
PciDeviceName	Returns a text string corresponding to the vendor and device IDs supplied as parameters.
PciClassName	Returns a text string corresponding to the class ID supplied as parameters.

B INtime software components

This appendix describes product components. The descriptions assume that the installation path is c:\Program Files\INntime\...:

Configuration option	Page
Blue.exe (Windows crash program)	132
BlueCon.rta	132
Clk0Jitr.rta	133
EventMsg.dll	133
INtimeConfiguration.exe	133
INtime.hlp	133
INscope.exe	135
INtex.exe	136
INtime Performance Monitor (INtmPerf.* files)	136
INtime RT Client Browser	136
ItWrpSrv.exe (Service wrapper)	137
Jitter.exe	137
LdRta.exe (INtime RT Application Loader)	137
LoadRtk.exe (INtime Kernel Loader)	138
MFC*.dll files	138
netstat.rta	139
NTX header files	139
NTX import libraries	139
NTX DLLs	139
NtxRemote2.exe (INtime Remote Connection Manager)	139
OvwGuide.pdf	140
Pdsproxy.dll files	140
Ping.rta	140
Project files	140
RT node files	141
RT header files	142
RT interface libraries	142
RT Stack Services	142
RtClkSrv.exe (INtime Clock Synchronization Service)	143
RtDrvrW5.awx (RT Device Driver wizard)	143
RtELServ.exe (INtime Event Log Service)	144
RtIf.sys (RT Interface Driver)	144
RtIOCons.exe (INtime I/O console)	145
RtIOSrv.exe (INtime I/O Service)	145
RtkImage.dbg (RT kernel with debug capabilities)	146
RtkImage (RT kernel)	146

Configuration option	Page
RtMgr.exe (RT Process Manager)	146
RtNdSrv.exe (INtime Node Detection Service)	147
RtPrcMgr.rta	147
RtProcW5.awx (RT Process wizard)	147
RtProcAddinW5.awx (RT Process Add-in wizard)	147
RtRegSrv.exe (INtime Registry Service)	148
RtRslWiz.awx (RT Shared Library wizard)	148
Soft-Scope auxiliary files	148
SSBug.exe (Soft-Scope record conversion)	148
SSWin32.exe (Soft-Scope RT debugger)	149
Stacksrv.exe (INtime Network Service)	149
StarveAlarm.exe (Windows Starvation Alarm)	150

Blue.exe (Windows crash program)

A Windows program that causes the Windows system to have a 'blue screen crash'. Use this program to validate the operation of the INtime software after Windows experiences a "blue screen crash".

Item	Description
Pathname	c:\Program Files\INtime\bin\blue.exe
Invocation	Invoke from the DOS box as follows: `blue -really`

BlueCon.rta

An INtime application that manages the system console and keyboard on a remote node. It sets up the RT "C" library's printf and scanf routines to interface with the console and keyboard, enabling communication between a remote INtime node and an operator at that node.

Item	Description
Pathname	c:\Program Files\INtime\remote\common\bluecon.rta
Invocation	Invoked by the Remote INtime Run-Time Application Loader when the system boots.

Clk0Jitr.rta

An RT application started by Jitter.exe. This application measures the minimum, maximum, and average times between low-level ticks via an Alarm Event Handler. For more information, see INtime Graphical Jitter.

Item	Description
Pathname	c:\Program Files\INtime\projects\jitterrt\clk0jitr.rta
Invocation	Jitter.exe loads this RT application as it starts.

EventMsg.dll

A resource DLL that associates an event ID with a message. You can add your own messages and event IDs to this DLL by using Microsoft Visual Studio on the project c:\Program Files\INtime\projects\eventmsg.

Item	Description
Pathname	c:\Program Files\INtime\system32\eventmsg.dll
Invocation	The INtime Event Log service loads this DLL at runtime.

INtimeConfiguration.exe

A Windows program that configures INtime software.

Item	Description
Pathname	c:\Program Files\INtime\bin\INtimeConfiguration.exe
Invocation	Double-click the icon associated with the file in the INtime program folder (Start>All Programs>INtime>INtime Configuration).

INtime.hlp

INtime software contains the following Help files:

- Main Help files
- Utility Help files
- Embedded C++ Help files

Main Help files

A Windows Help file that describes INtime software.

The Help file uses the file, c:\Program Files\INtime\help\intime.cnt, to display INtime software information.

Item	Description
Pathname	c:\Program Files\INtime\help\intime.hlp
Invocation	Double-click the icon associated with the help file in the INtime program folder (Start>All Programs>INtime>INtime Help).

Utility Help files

Windows Help files that describe INtime software's utilities. Utilities include:

Utility	Help files
Configuration utility	INtmCnfg.cnt INtmCnfg.hlp
RT Process wizard	RtProcW.cnt RtProcW.hlp
RT Process Add-in wizard	RtAddinW.cnt RtAddinW.hlp
RT Device Driver wizard	RtDrvrW.cnt RtDrvrW.hlp
RT Process Manager	RTMgr.cnt RTMgr.hlp
RT Application Loader	LdRta.hlp
RT Service Wrapper	ITWrpSrv.hlp
Starvation alarm	StrvAlrm.cnt StrvAlrm.hlp

Embedded C++ Help files

HTML and GIF files that describe Embedded C++ calls and syntax. Files include:

Pathname	Files			
c:\Program Files\ INtime\help\ecpp\	_index.html	cstdlib.html	iosfwd.html	setjmp.html
	assert.html	cstring.html	iostrea2.html	signal.html
	cassert.html	ctime.html	iostream.html	sstream.html
	cctype.html	ctype.gif	istream.html	stdarg.html
	cerrno.html	ctype.html	lib_cpp.html	stddef.html
	cfloat.html	errno.html	lib_file.html	stdexcep.html
	charset.html	escape.gif	lib_over.html	stdio.html
	climits.html	exceptio.html	lib_prin.html	stdlib.html
	clocale.html	express.html	lib_scan.html	stream.gif
	cmath.html	float.html	limits.html	streambu.html
	complex.html	format.gif	locale.html	string.html
	crit_pb.html	fstream.html	math.html	string2.html
	crit_pjp.html	fstream2.html	new.html	strstrea.html
	csetjmp.html	function.html	new2.html	strtod.gif
	csignal.html	index.html	ostream.html	strtol.gif
	cstdarg.html	iomanip.html	preproc.html	time.gif
	cstddef.html	iomanip2.html	print.gif	time html
	cstdio.html	ios.html	scan.gif	

Item	Description
Pathname	c:\Program Files\INtime\help\ecpp_index.html
Invocation	Do one of these: • Access INtime software Help, then select Using INtime software>Other system calls>EC++ calls. • Double-click the _index.html file in the EC++ help folder.

INscope.exe

A Windows program that uses NTX calls to communicate with its self-loaded RT counterpart to trace execution of INtime applications.

Item	Description
Pathname	c:\Program Files\INtime\bin\inscope.exe
Invocation	Double-click the icon associated with the file in the INtime Program folder (Start>All Programs>INtime>INtime Real-time Performance Analyzer).

INtex.exe

A Windows application which allows you to browse the objects in an INtime kernel.

Item	Description
Pathname	c:\Program Files\INtime\bin\intex.exe
Invocation	Double-click the icon associated with the file in the INtime Program folder (Start>All Programs>INtime>INtime Explorer).

INtime Performance Monitor (INtmPerf.* files)

The INtime Performance Monitor reports INtime Kernel CPU usage to the Windows Performance Monitor.

INtmPerf.ini is a setup file required as a part of the INtime installation process. It is used by the Windows LOADCTR utility to add the proper registry keys and settings.

Item	Description
Pathname	c:\Program Files\INtime\system\intimperf.dll c:\Program Files\INtime\system\intimperf.ini

INtime RT Client Browser

An ActiveX control that you can add to your INtime applications. For information about adding this browser to INtime applications, see *Adding the INtime RT Client Browser to your INtime application* on page 88.

Item	Description
Pathname	c:\Program Files\INtime\system32\inbrow.ocx

You can also add the INtime RT Client Browser using the NTX API call ntxFindINtimeNode and its associated export library and DLL, See INtime Help for detailed information about this call.

Item	Description
Pathname	c:\Program Files\INtime\nt\lib\ntxrow.lib c:\Program Files\INtime\bin\ntxrow.dll

ItWrpSrv.exe (Service wrapper)

A Windows program that installs and removes a user application (either Windows or INtime software) as a Windows service.

Item	Description
Pathname	c:\Program Files\INtime\bin\itwrpsrv.exe
Invocation	A user runs this program to install a custom service to Windows. Run it with the `-help` switch to display usage parameters. Once installed, run ItWrpSrv.exe using the -remove switch to remove the custom Windows service.

Jitter.exe

A Windows program that automatically starts Clk0Jitr.rta, then processes output and displays a histogram. This application measures the minimum, maximum, and average times between low-level ticks via an Alarm Event Handler. For more information, see INtime Graphical Jitter.

Item	Description
Pathname	c:\Program Files\INtime\projects\jitternt\jitter.exe
Invocation	Select Start>All Programs>INtime>INtime Graphical Jitter

LdRta.exe (INtime RT Application Loader)

A Windows program that loads and starts the RT portion of INtime applications. The loader has two parts: the program that executes on Windows, and an RT "helper" process that performs the RT portion of a load operation. The "helper" process is part of the RT kernel.

The Windows-resident portion of the RT Application Loader is a 32-bit Windows program that uses NTX library calls to load and start INtime applications under the RT kernel.

The RT Application Loader:

- Supports loading of both 32-bit Microsoft PE code (the output of Developer Studio) and 32-bit OMF386 code.
- Supports both command line and dialog-based operation. Supports specification of the file to load, optional debug arguments, and optional program arguments.
- Recognizes the file extension ".RTA" (for RT application).

The INtime Installation processes set up a file association so that an INtime application loads automatically when a user double-clicks the file name in a supporting Windows application (such as Windows Explorer). If a default node

has not been established, such an invocation (double-click of the filename) displays ldrta.exe's user interface so you can establish a default node.

Item	Description
Pathname	c:\Program Files\INtime\bin\ldrta.exe
Invocation	Do one of these: • Double-click the icon associated with LdRta.exe (INtime RT Application Loader) in the INtime program folder (Start\All Programs\Intime\INtime Loader), then select the INtime application you want to load. • Double-click an INtime application executable which has a .RTA extension.

LoadRtk.exe (INtime Kernel Loader)

A 32-bit Windows program that loads the RT kernel after Windows starts. When set to automatically start, the Windows service manager launches the INtime Kernel Loader at system startup. In this case, the loader loads the RT kernel after the Windows kernel and after other Windows services, but before users log on. Otherwise, the INtime Kernel Loader is started manually using the Services applet (Start>Control Panel>Administrative Tools>Services).

The INtime Kernel Loader cooperates with the RT Interface Driver (RtIf.sys) to load the RT kernel image set up by the INtime Configuration Utility.

First it loads the specified image into the memory allocated by the RT Interface Driver, then it makes a request to the RT Interface Driver to start the RT kernel.

You can configure the RT kernel to start automatically at boot time by using the Windows Services Manager.

Item	Description
Pathname	c:\Program Files\INtime\bin\loadrtk.exe
Invocation	Start INtime software's RT kernel either in Manual or Automatic mode, using the Windows Services Manager (Start>Control Panel>Administrative Tools>Services>INtime Kernel Loader).

MFC*.dll files

Microsoft DLLs required by MFC programs; included in the event they were not installed with Windows.

Item	Description
Pathname	c:\Program Files\INtime\system32\mfc30.dll c:\Program Files\INtime\system32\mfc30d.dll c:\Program Files\INtime\system32\mfc42.dll

netstat.rta

An RT application that displays statistics about the INtime TCP/IP stack.

Item	Description
Pathname	c:\Program Files\INtime\bin\netstat.rta
Invocation	netstat [-ains] [-p protocol] [interval]

NTX header files

Pathname	Files
c:\Program Files\ INtime\nt\include\...	ntx.h ..... C NTX header file

NTX import libraries

Pathname	Files
c:\Program Files \INtime\nt\lib\...	ntx.lib.....NTX import library ntxext.lib .....Extended NTX import library.

NTX DLLs

DLLs provided with INtime software used by Windows applications to communicate with INtime applications using NTX system calls.

Item	Description
Pathname	c:\windows\system32\ntx.dll c:\windows\system2\ntxext.dll
Invocation	Windows loads these DLLs when a Windows application attempts the first NTX call.

NtxRemote2.exe (INtime Remote Connection Manager)

Manages connections with remote INtime nodes. Runs as the INtime Remote Connection Manager.

Item	Description
Pathname	c:\Program Files\INtime\bin\ntxremote2.exe
Invocation	Start INtime software's ntxremote2.exe either in Manual or Automatic mode using the Windows Services Manager (Start>Control Panel>Administrative Tools>Services>INtime Remote Connection Manager).

OvwGuide.pdf

The INtime Software Overview Guide in PDF format. This file requires Acrobat32.exe.

Item	Description
Pathname	c:\Program Files\INtime\help\OvwGuide.pdf
Invocation	Double-click the file in the Windows Explorer. This launches Acrobat32.exe which, in turn, opens the OvwGuide.pdf file for viewing.

Pdsproxy.dll files

Performance data server DLL. Used by the INtime Starvation Alarm to gather INtime CPU usage information.

Item	Description
Pathname	c:\Program Files\INtime\system32\
Invocation	Loaded by the INtime Starvation Alarm when it first starts.

Ping.rta

An RT application that you use to test the network connection between two machines. If connected, the remote system responds with a series of packet echoes.

Item	Description
Pathname	c:\Program Files\intime\bin
Invocation	Start the program using the RT Application Loader. Use this syntax: `ping` *`TargetSystemIPAddress`*

Project files

INtime software has a number of sample applications that you can use as samples for your own INtime applications.

The source code for these applications is provided in Microsoft Developer Studio version 6.0 generation format, and reside in separate directories. For example, the INtime API test program source files reside in the c:\Program Files\INtime\Projects\RTTest directory.

Click on the desired project's name (Start>All Programs/INtime/Sample Code and Projects) to open the Microsoft Developer Studio and find out more about each of these projects:

- INtime API Sample
- INtime Serial Driver Sample
- INtime Graphical Jitter
- Real-time Interrupt Sample
- C and C++ Samples for Debugger
- TCPIP>TCIP-IP Client Sample
- TCPIP>TCP-IP Server Sample
- Fault Handling (ntrobust)
- Floating Point Exception Handling
- RSL Example
- NTX Sample (MsgBoxDemo)
- INtime Windows STOP Detection sample (STOPmgr)
- INtime USB Client sample

Item	Description
Pathname	c:\Program Files\INtime\projects\<Sample Directory>\for source code c:\Program Files\INtime\projects\<Sample Directory>\ debug for executables
Invocation	Use Microsoft Developer Studio to edit/compile these sample applications. Use the INtime RT Application Loader to load the resulting sample application executables.

RT node files

Contains the files required to build an RT node. For more information about building RT nodes, see *Chapter 8, Preparing an RT node*.

Pathname	Files
c:\Program Files\INtime\remote\common	

RT header files

Pathname	Files
c:\Program Files\INtime\rt\include\...	C RT and C library header files.
c:\Program Files\INtime\rt\include\sys\...	Additional C RT and C library header files.
c:\Program Files\INtime\rt\include\arpa\...	Internet C header files.
c:\Program Files\INtime\rt\include\netinet\...	Additional internet C header files.
c:\Program Files\INtime\rt\include\services\...	C RT Services header files.

RT interface libraries

Pathname	Files	
c:\Program Files\ INtime\rt\lib\...	ciff3m.lib	CLIB function interface library
	ecpp.lib	Embedded C++ library
	net3m.lib	Sockets utility flat library
	pcibus.lib	PCI library
	rmxiff3m.lib	Flat RMX library
	rt.lib	INtime API interface library
	rtpplib	Embedded C++ library
	rtppd.lib	Embedded C++ debug library
	rtserv.lib	INtime port interface library
	usbss.lib	INtime USB subsystem interface library

RT Stack Services

INtime RT components that make up the RT TCP/IP Stack. These components include NIC drivers and TCP/IP Stack Layers.

Pathname	Files	
c:\Program Files\ INtime\remote\common\...	3c59x.rta	INtime 3COM Driver
	eepro100.rta	INtime EEpro 100 Driver
	ip.rta	INtime IP
	loopback.rta	INtime Loopback Driver
	ne.rta	INtime NE Driver
	rip.rta	INtime Raw IP
	rtl8139.rta	Realtek Driver
	tcp.rta	INtime TCP
	udp.rta	INtime UDP

RT USB Interface Drivers

INtime RT components that make up the RT USB Subsystem. These components include Host Controller drivers for UHCI, OHCI, and EHCI (USB 2.0) interfaces.

Pathname	Files
c:\Program Files\INtime\bin	usbss.rsl .....INtime USB Subsystem Shared Library uhci.rta .....INtime USB Universal Host Controller Interface Driver ohci.rta .....INtime USB Open Host Controller Interface Driver ehci.rta .....INtime USB Enhanced Host Controller Interface (USB 2.0) Driver

RtClkSrv.exe (INtime Clock Synchronization Service)

A Windows program that provides time-of-day and time interval services used to synchronize the RT time-of-day clock to the Windows time-of-day clock.

You can have the INtime Clock Synchronization Service start automatically at boot time by using the Windows Services Manager to set up the INtime Clock Synchronization Service for Automatic Startup. If the INtime Node Detection Service has not yet started, it is automatically started by the INtime Clock Synchronization Service.

Item	Description
Pathname	c:\Program Files\INtime\bin\rtclksrv.exe
Invocation	This program is run as a Windows service (Start>Control Panel>Administrative Tools>Services> INtime Clock Synchronization Service).

RtDrvrW5.awx (RT Device Driver wizard)

An MSVC 6.0 Application Wizard that you use to develop device drivers for INtime applications.

Item	Description
Pathname	c:\Program Files\INtime\msdev\template\rtdrvrw5.awx
Invocation	Invoked by the MSVC 6.0 IDE.

RtELServ.exe (INtime Event Log Service)

A 32-bit Windows program that supports Windows event log manipulation by RT threads. The RT Event Log Service receives, processes, and responds to requests from the RT application library's event log entry points. When the RT Event Log Service receives a request, it blocks the calling thread, executes the appropriate Win32 event log function, and replies to the original request (unblocking the calling thread). If the Windows host and/or the RT Event Log Service terminates execution, it terminates all pending requests with an E_EXIST error.

The RT Event Log Service supports a single request: to write an entry from the RT client event source at the end of the local PC's Application event log.

Neither the RT application library nor the RT Event Log Service support event logging to the system or security event logs, event logging by a source other than RT client, event logging to a remote PC, or backing up an event log file.

You can have the INtime Event Log Service start automatically at boot time by using the Windows Services Manager to set up the INtime Event Log Service for Automatic Startup. If the INtime Node Detection Service has not yet started, it is automatically started by the INtime Event Log Service

Item	Description
Pathname	c:\Program Files\INtime\bin\rtelserv.exe
Invocation	Start the INtime Event Log Service using the Windows Services Manager (Start>Control Panel>Administrative Tools>Services).

RtIf.sys (RT Interface Driver)

A Windows kernel mode device driver that co-manages the OSEM used by the INtime product to add RT capabilities to Windows. This driver:

- Establishes the initial RT kernel environment.
- Co-manages switching between the Windows and INtime runtime environments.
- Supports communication and synchronization mechanisms between Windows threads and RT threads by relaying NTX library requests.

Note: For detailed information, see Help.

Item	Description
Pathname	c:\windows\system32\drivers\rtif.sys
Invocation	Windows loads this driver at system initialization time.

RtIOCons.exe (INtime I/O console)

The RT I/O Console is a 32-bit Windows program that provides support for Windows console I/O for RT threads. It creates and manages a single console window and executes keyboard data (input) requests and display (output) requests from the RT thread.

Item	Description
Pathname	c:\Program Files\INtime\bin\rtiocons.exe
Invocation	Invoked by RtIOSrv.exe (INtime I/O Service)

RtIOSrv.exe (INtime I/O Service)

A 32-bit Windows program that provides Windows file system and console I/O support for RT threads. This program acts as a server to RT "C" library to obtain a console window for display (via printf) of application data and to receive (via scanf) user input from the system keyboard.

When the RT "C" library receives an stdio request from a real-time thread, it checks to see if a console exists. If a console exists, it relays the request to the appropriate RT I/O console If no console exists, it blocks the thread making the request, creates an RT I/O console window for the thread via the INtime I/O Service, and relays the request to the RT I/O console. When a request completes, the INtime I/O Service unblocks the corresponding thread and relays the reply. If Windows terminates execution, the RT I/O Console terminates all pending requests with an E_EXIST error.

When the RT "C" library receives a file I/O request from an RT thread, it blocks the thread making the request and forwards the request to the INtime I/O Service for completion. When a request completes, the RT "C" library unblocks the thread making the request and relays the reply to the thread.

Use of the INtime I/O Service is restricted to the RT "C" library.

You can have the INtime I/O Service start automatically at boot time by using the Windows Services Manager to set up the INtime I/O Service for Automatic Startup. If the INtimeNodeDetection Server has not yet started, the INtimeIO Server starts it automatically.

Item	Description
Pathname	c:\Program Files\INtime\bin\rtioserv.exe
Invocation	Start the INtimeIO Server using the Windows Services Manager (Start>Control Panel>Administrative Tools>Services).

RtkImage.dbg (RT kernel with debug capabilities)

The RT kernel binary image with debug capabilities, loaded by LoadRtk.exe (INtime Kernel Loader).

This is identical to RtkImage (RT kernel), except for debug capabilities. When you start the INtime Kernel Loader service, the RT kernel program listed in the registry (RTKIMAGE.DBG or RTKIMAGE) loads and starts.You identify which program to use during installation, and you can change it by using the INtime Configuration utility.

Item	Description
Pathname	c:\Program Files\INtime\rtkimage.dbg
Invocation	Launched by LoadRtk.exe (INtime Kernel Loader).

RtkImage (RT kernel)

The RT kernel binary image without debug capabilities, loaded by LoadRtk.exe (INtime Kernel Loader).

This is identical to RtkImage.dbg (RT kernel with debug capabilities), except for debug capabilities. When you start the INtime Kernel Loader service, the RT kernel program listed in the registry (RTKIMAGE.DBG or RTKIMAGE) loads and starts.You identify which program to use during installation, and you can change it by using the INtime Configuration utility.

Item	Description
Pathname	c:\Program Files\INtime\rtkimage.exe
Invocation	Launched by LoadRtk.exe (INtime Kernel Loader).

RtMgr.exe (RT Process Manager)

A 32-bit Windows program that you use to monitor and manage RT processes. It is similar in visual appearance and functionality to the Windows task manager, but is specific to RT processes and threads.

Item	Description
Pathname	c:\Program Files\INtime\bin\rtmgr.exe
Invocation	Double-click the icon associated with the file in the INtime Program folder (Start>All Programs>INtime>Realtime Process Manager).

RtNdSrv.exe (INtime Node Detection Service)

A 32-bit Windows program that detects RT clients, both local and remote. This program checks for and registers RT clients that exist in both of these locations:

- RT clients configured in INtime Configuration utility.
- RT clients available to the system.

Item	Description
Pathname	c:\Program Files\INtime\bin\rtndsrv.exe
Invocation	Start the INtime Node Detection Service using the Windows Services Manager (Start>Control Panel>>Administrative Tools>Services).

RtPrcMgr.rta

An INtime application loaded by the RT Process Manager. It is the RT server which provides RT data for the RT Process Manager.

Item	Description
Pathname	c:\Program Files\INtime\bin\rtprcmgr.rta
Invocation	Invoked by RtMgr.exe (the RT Process Manager).

RtProcW5.awx (RT Process wizard)

An MSVC 6.0 Application Wizard that you use to develop the RT portion of INtime applications.

Item	Description
Pathname	c:\Program Files\INtime\msdev\template\rtprocw5.awx
Invocation	Invoked by the MSVC 6.0 IDE.

RtProcAddinW5.awx (RT Process Add-in wizard)

An Application wizard for Microsoft Developer Studio, version 6.0, that you use to add supplemental files to the already-generated RT portion of INtime applications.

Item	Description
Pathname	c:\Program Files\INtime\msdev\addins\rtprocaddinw5.awx
Invocation	Invoked by the MSVC 6.0 IDE.

RtRegSrv.exe (INtime Registry Service)

A Windows program that provides RT Registry Clients access to the Windows registry.

Item	Description
Pathname	c:\Program Files\INtime\bin\rtregsrv.exe
Invocation	Start the INtime Registry Service using the Windows Services Manager (Start>Control Panel>Administrative Tools>Services).

RtRslWiz.awx (RT Shared Library wizard)

An MSVC 6.0 Application Wizard that you use to develop a Realtime Shared Library (RSL) for an INtime application.

Item	Description
Pathname	c:\Program Files\INtime\msdev\template\rtrslwiz.awx
Invocation	Invoked by the MSVC 6.0 IDE.

Soft-Scope auxiliary files

Macro/configuration files used by Soft-Scope to provide RT kernel-aware, source level debugging capabilities.

Item	Description	
Pathname	c:\Program Files\INtime\sswin32\...	
Files	SDBWin.MAC	SSWin32.INI
	SSWin.MACS	SSWin32.CNT
	SSWin32.HLP	

SSBug.exe (Soft-Scope record conversion)

A 32-bit Windows program that converts debug records from Microsoft OMF to OMF386 format. Soft-Scope requires debug records in OMF386 format. Soft-Scope automatically executes SSBug.exe when it loads an executable containing debug records in Microsoft OMF.

Item	Description
Pathname	c:\Program Files\INtime\sswin32\ssbug.exe
Invocation	Invoked by SSWin32.EXE

SSWin32.exe (Soft-Scope RT debugger)

A 32-bit Windows program that communicates via NTX calls to the RT kernel to provide source level, multi-tasking debug capabilities for INtime applications.

Two methods of debugging are possible using SoftScope: one where the RT process is loaded by Soft-Scope (via the RT Application Loader), and another where the RT process is loaded by the user via the RT Application Loader and connected to by Soft-Scope. Both methods requires the RT Application Loader to insert an "INT 3" instruction at the entry point of the loaded application's initial thread. Once the breakpoint is reached, the thread is suspended until resumed by Soft-Scope. Soft-Scope loads the RT process's symbols and picks up debugging the thread at its entry point.

Item	Description
Pathname	c:\Program Files\INtime\sswin32\sswin32.exe
Invocation	Double-click the icon associated with SSWin32.exe in the INtime program folder (Start>All Programs>INtime>Soft-Scope Real-time Application Debugger).

Stacksrv.exe (INtime Network Service)

A Windows program that starts the INtime TCP/IP Stack components and NIC Drivers.

Item	Description
Pathname	c:\Program Files\INtime\bin\stacksrv.exe
Invocation	Start the INtime Network Service using the Windows Services Manager (Start>Control Panel>Administrative Tools>Services).

StarveAlarm.exe (Windows Starvation Alarm)

A 32-bit Windows program that detects starvation based on how many CPU cycles Windows receives within a specified time interval. You specify both the cycles and the time interval; the Starvation Alarm does the detective work.

When starvation occurs, the Starvation Alarm utility:

- Updates the Windows event log with the time and type of starvation event.

  The log receives entries for these occurrences of each starvation specification: 1, 10, 100, 1000. Entry 1000 also indicates that no further entries will be made until the application exits. Upon application exit, the log receives an entry for each starvation specification that occurred, with the total number of occurrences since the last associated entry provided in the entry.

- Emits the alarm type you specify, an audible beep, a visible message, or both.
- Changes the appearance of the icon in the task bar to indicate that starvation occurred.

Item	Description
Pathname	c:\Program Files\INtime\bin\starvealarm.exe
Invocation	Double-click the icon associated with StarveAlarm.exe in the INtime program folder (Start>All Programs>INtime>Starvation Alarm).

C Adding INtime software to an XP Embedded configuration

You can add INtime components to your XP Embedded configuration using the standard mechanism provided in the XP Embedded Developer's Kit.

The INtime.sld file (c:\Program Files\INtime\Xpembedded\intime.sld) defines the INTime components and should be imported into the component database using the Component Database Manager. After creating new target XPE images, you can view and add INtime components from the Database.

INtime components are found in the component hierarchy under this key:

```
Software\System\OEM System Extensions\Infrastructure\
```

D Troubleshooting

This appendix lists problems you may encounter while running INtime software, and explains how to avoid or resolve those problems.

Complete these steps to resolve INtime software problems:

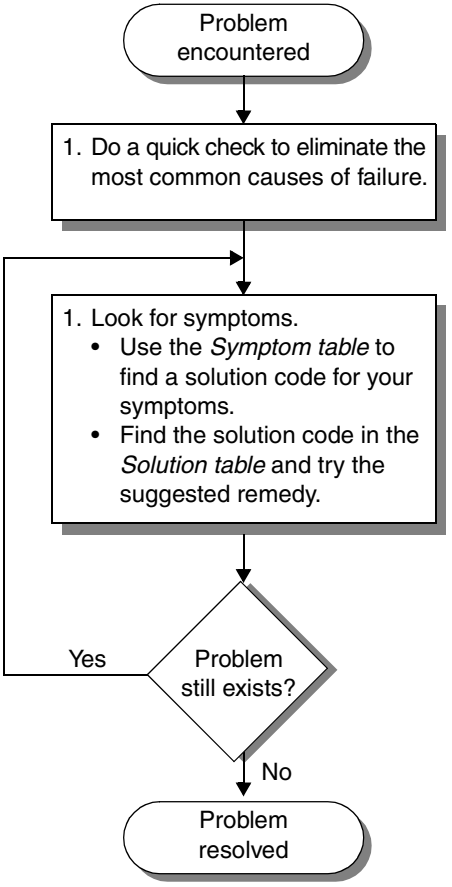

Problem encountered
1. Do a quick check to eliminate the most common causes of failure.
1. Look for symptoms.
• Use the *Symptom table* to find a solution code for your symptoms.
• Find the solution code in the *Solution table* and try the suggested remedy.
Problem still exists?
Yes
No
Problem resolved

Figure D-1. Troubleshooting INtime software problems

1. Do a quick check

Many problems are solved by correcting these:

- **Is the RT kernel running?** You must start the RT kernel prior to starting most INtime development tools and all INtime applications.
- **Did you use the INtime RT Application wizard to set up the project?** Using this wizard automatically starts an INtime project and ensures that the project includes proper settings.

2. Look for symptoms

Scan the *Symptom table* until you find a symptom that your system exhibits. The Symptom table lists solution codes in the order most likely to occur.

Locate the corresponding solution codes in the *Solution table* and take the recommended action. You may need to try several solutions before you successfully resolve the problem.

Table D-1. Symptom table

Symptom	Solution codes
INtime software does not install.	1, 2, 3, 4, 5, 6
Tools in the INtime development environment do not run.	11
The INtime application reports build errors.	9, 10
The serial mouse does not respond after you start the RT kernel.	12
The INtime application does not load.	11, 13
The file is not recognized as a valid INtime application.	14
The INtime application terminates unexpectedly.	15
The INtime application displays this message: `Running out of Virtual Address Space (error code 0xf0)`	15
The INtime application reports an unsupported C library function.	10
The INtime application performs terminal I/O, and no terminal window appears.	17
Only a blue screen displays.	20
Windows window events and applications respond slowly.	19, 16
The system doesn't respond to attempts to move the cursor or make entries from the keyboard.	18, 20, 19, 16, 8
You're running SDM and neither the system console nor the SDM terminal respond to attempts to move the cursor or make entries from the keyboard.	7
While running Soft-Scope, you tried to suspend or resume threads not visible in the Threads windows and the INtime application stopped running.	21

Table D-2. Solution table

Installing INtime software	1	Adjust your system configuration. Your system has an incompatible system configuration.
	2	Install INtime on a Pentium system that has only a single processor. INtime software does not install or run on systems that have more than one processor, such as a dual-Pentium system. **Note**: You can install INtime on a dual-Pentium enabled system that has only one processor. For information about installing on such a system, select Start>All Programs>INtime>INtime v2.20 Release Notes.
	3	Ensure that your PC has: • A minimum 20MB of DRAM. • A minimum 10MB disk space • Windows NT, version 4.0 or later.
	4	Ensure that you exit all programs before you install INtime software.
	5	Ensure that your system does not contain a previous version of INtime. If a previous version exists, uninstall it by selecting Start>Control Panel>Add/Remove programs.
	6	Ensure that you are logged on with Administrator privileges.
Developing INtime applications	7	Use the SDM PDP <logical address> command to get the physical address of memory in question, followed by S c80:physical address (returned by PDP). Do not use the SDM D command to display Ring 3 code/data in a Virtual Segment at an address not populated with memory. Do not use the SDM S command to modify memory using a Ring 3 code segment (read-only).
	8	If you're running SDM, no action is required. When you run SDM, you access the INtime software via the SDM terminal and the Windows system console no longer responds.
	9	Verify project build settings and ensure that you: • Select Not Using MFC to disable use of MFC classes. • Include the <INtime install path>\rt\include directory as a preprocessor directive. • Ignore all default libraries as a link option.
	10	Remove C library calls not supported by INtime. For a list of C library calls that INtime supports, run INtime Help and select Programmer's reference>C library reference.

Table D-2. Solution table

Loading INtime applications	11	Start the RT kernel before starting an INtime application. For information about starting the RT kernel, see *Chapter 9, Operation*.
	12	Do one of these: • If your mouse is on COM2, switch it to COM1. • Switch the RT kernel debug port to COM1 or disable RT kernel debug.
	13	Run the RT Application Loader and select the Advanced option to increase the amount of virtual address space in 4 MByte increments. Not enough memory exists to load the application. The program size and its heap and stack requirements specified when it was built may exceed the amount of memory available for INtime applications. For more information about changing memory options for loading INtime applications, see Help in the RT Application Loader.
	14	Rebuild the project. You may have tried to run a corrupted executable. This can occur, as an example, when a project is open and Windows halts.

Table D-2. Solution table

Running INtime applications	15	Run the RT Process Manager to obtain debug information. For information about running the RT Process Manager, see *After you start INtime software* in *Chapter 9, Operation*.
	16	Set the RT kernel tick interval above 200usec. When the kernel tick drops below this rate, Windows slows down because the CPU devotes too much time to switching between the Windows and INtime kernels.
	17	Ensure that the INtime I/O Service is running. INtime applications that perform terminal I/O require that the INtime I/O Service runs. For information about starting this service, see *Starting the RT kernel and related components* in *Chapter 9, Operation*.
	18	End the INtime application: 1. Invoke the INtime Explorer (Start>All Programs>INtime>INtime Explorer). 2. Highlight the INtime application you want to terminate. 3. Right click the mouse. A pulldown menu displays. 4. Select Delete. INtex prompts you to confim the deletion process. 5. Select Yes. If you cannot display the INtime Explorer (i.e., the Windows screen seems frozen and the mouse does not respond), the INtime application may have halted or may be monopolizing system resources. If you suspect the latter, see solution 19. **Note**: To verify whether Windows has halted, try to access the file system from another system. If you can, Windows is still running but cannot respond.
	19	Adjust your INtime application so that the RT portion of your INtime applications do not dominate CPU time. Your INtime application may be designed to monopolize too many system resources. For more information about designing applications to balance RT and Windows activity, see *Methodology* in *Chapter 5, Designing RT applications*. **Note**: To aid in diagnosing system resource misuse, try to access the file system from another system. If you can, Windows is still running but cannot respond.

Table D-2. Solution table

Running INtime applications (continued)	20	Exit INtime applications, if possible, then reboot your system: One of these situations may have occured: • Windows halted. • A Windows application halted. • An INtime application monopolized system resources. **Note**: Try other solutions before trying this one.
	21	Use the Command Line (Control+L) Suspend <thread handle> command on application threads identified via SDB commands under Soft-Scope. While Soft-Scope was running, you tried to suspend or resume threads not visible in the Threads window.

Other resources

If the information in this chapter doesn't solve the problem, you may want to contact TenAsys as described in *Where to get more information* on page v.

Soft-Scope hints & tips

Ensure that your application contains no free-running threads. Every thread should either wait for an event, or should sleep on occasion so that Windows, and thus Soft-Scope, can run.

If Soft-Scope terminates abnormally, it can corrupt or delete its .INI file (SSWIN32.INI). A backup SSWIN32.INI file is installed in <INtime Install Path>\backup directory. If Soft-Scope signs on with no loading of macros, replace the SSWIN32.INI in <INtime Install Path>\sswin32 directory with a copy of the one in the <INtime Install Path>\backup directory.

If a debug session terminates abnormally. Soft-Scope can sometimes get out of sync with it's self-loaded RT counterpart (SskNw.TMP). If this occurs, use INtime Explorer to delete any child processes of the SskNew.TMP process on the INtime System, and then delete SskNew.TMP itself. Soft-Scope will now load a new version of SskNew.TMP, and the two will be in Sync again.

Glossary

application loader	The layer of an INtime application that loads programs into memory for execution, such as when a user enters a command at the console.
asynchronous	Events that occur at random times.
BIOS	(Basic I/O System) On a PC system, the code that resides in ROM to supply OS-independent access to the computer's I/O system.
BSOD	(Blue Screen of Death) An acronym used to describe total Windows failure.
client	On a network, a client is a computer which makes requests of a remote system that acts as a server. For example, a client could request a remote server to supply it with data from one of the server's disk files.
descriptor	An 8-byte data structure taken from a descriptor table in memory. Descriptors provide the CPU with the data it needs to map a logical address into a linear address. The fields of a descriptor include information concerning segment size, base address, access rights, and segment type (such as read/write segment, executable segment, call gate, task gate, trap gate, etc).
determinism	Predictable response time. Enables threads to execute before their deadlines expire.
device controller	The hardware interface between the CPU or system bus and a device unit.
device driver	The software interface between the I/O system and a device controller.
Distributed INtime	A configuration of INtime where an INtime kernel (RT client) runs on a CPU that does not run Windows.
encapsulation	A characteristic of object-based systems. The representation of an object is hidden from the user of that object. Only the object's type manager can manipulate an object. Users of an object can manipulate the object only by invoking type manager functions for the object.
EOI	(End of Interrupt) A command sent to a PIC to indicate that an interrupt handler completed processing an interrupt.
event-driven	Applications can respond to interrupts as they occur; they do not waste time polling for interrupts.
exception handler	A program that receives control when either the operating system or hardware detects an error.
exchange object	Generic name for object types managed by INtime software that allow threads to synchronize and communicate with each other. Exchange objects include: semaphores, mailboxes, and regions.

execution state Thread state. Thread execution states include: running, ready, asleep, suspended, or asleep-suspended.

FIFO First in, first out.

GDT (Global Descriptor Table) A memory segment that contains descriptors for code, data, and descriptor table segments.

handle An object identifier.

IDT Interrupt descriptor table.

interrupt A signal from a device such as a NIC (Network Interface Card) or IDE hard disk controller. You connect interrupt sources to the processor through a PIC (Programmable Interrupt Controller).

interrupt handler Code executed first in response to a hardware interrupt. This code runs in the context of the thread that was running when the interrupt occurred. Interrupt handlers must save the current context on the stack before using any CPU registers.

interrupt levels PICs manage interrupts by presenting them to the system processor as discreet levels in priority order. INtime software handles more critical interrupts first, and keeps track of which interrupts occurred, the order in which they occurred, and which ones have not been handled.

interrupt response time The time between a physical interrupt happening and the system beginning to execute the interrupt handler. By being able to calculate a predictable worst-case response time to interrupt processing, a real-time system can be designed to ensure that incoming data is handled before it becomes invalid.

INtime node A computer that simultaneously executes both a Windows host and an RT client that are connected via the OSEM.

MAC (Media Access Control) The 6-byte Ethernet address for a computer node on a network.

mailbox An RT kernel object type managed by the RT kernel. Mailboxes are used for interthread synchronization and communication. Two types of mailboxes exist:

- **Data mailbox**: sends and receives data. Available in both high and low level.
- **Object mailbox**: sends and receives object handles. Available only in high level.

memory address The architectural mechanism used by x86 processors to access an individual byte of system physical memory. In the 32 bit protected mode environment of Windows and INtime, memory addresses are 32 bit linear addresses relative to a 16 bit descriptor that is loaded into a CPU segment register. The INtime kernel manages the 16 bit descriptors (virtual segments - VSEGs) for INtime applications that only use 32 bit linear addresses to access code and data (flat model applications generated with MSVC tools).

memory area Provides memory for threads to use for many purposes, including communicating and storing data.

memory pool An amount of memory, with a specified minimum and maximum, allocated to a process. The basis of INtime software's memory management. The initial memory pool is all the memory available to the application (that is, free space memory). It is managed by the OS and allocated to the application on request.

message port An RT kernel object type managed by the kernel. Used to provide an access point for an INtime application thread to communicate with an INtime service.

multi-programming Ability of an operating system to simultaneously run several unrelated applications on a single system. Allows more than one application to run at a time.

multithreading Ability of an operating system to run multiple threads at virtually the same time. When the operating system stops executing one thread, it resumes/starts executing another thread. This transition from one thread to another is called a thread switch.

object An instance of a data structure that can be accessed only through a set of functions provided by a type manager.

object directory A storage area within an INtime process where objects can be cataloged, i.e. have a name associated with the objects so that other theads may refer to/access the cataloged object by name.

PIC (Programmable Interrupt Controller) An integrated circuit that can resolve simultaneous interrupt requests and negotiate a sequence of CPU interrupt requests based on the priorities of the requesting device controllers.

priority-based scheduling Abiltiy of an operating system to assign execution order importance values (priority) to each thread in the system. In an INtime system, the scheduling policy enforced by the INtime kernel is that the highest priority ready thread is/will immediately become the running thread. Thus, when thread A is running, if thread B becomes ready through some system event, and thread B is higher priority than thread A, then the INtime kernel will immediately preempt thread A, and make thread B the running thread.

priority inversion A situation in which a high-priority thread is effectively prevented from running by a lower-priority thread. Proper use of the RT kernel's region objects eliminates this problem.

process An RT kernel object type. Processes have memory pools and contain/own execution threads and other objects.

region An RT kernel object type managed by the kernel. Regions are binary semaphores with special suspension, deletion, and priority-adjustment features. You can use regions to provide mutual exclusion for resources or data.

round-robin scheduling A scheduling method where equal priority threads take turns running. Each thread gets a time slice, an equal portion of the processor's time.

RT	Real-time.
RT client	An RT subsystem designated to consume INtime services (booting, configuration, file system proxy, etc.) provided by a Windows host.
RT kernel	(Real-time kernel) An operating system that provides full RT functionality.
RT subsystem	A self-contained collection of software containing an RT kernel, a number of RT support processes, and the RT portion of zero or more INtime applications.
semaphore	An RT kernel object type managed by the kernel. Semaphores are used for interthread synchronization. A counter that takes positive integer values. Threads use semaphores for synchronization by sending units to and receiving units from the semaphores.
server	On a network, a server is any computer which responds to requests from remote systems. For example, file or print servers allow remote systems to access local disks or printers.
system call	A subroutine supplied by INtime software to provide a service, such as I/O processing or memory allocation. A programmatic interface you use to manipulate objects or control the computer's actions.
thread	An RT kernel object type managed by the RT kernel. Threads, or threads of execution, are the active, code-executing objects in a system.
thread switch time	The time required to save the context (data registers, stack and execution pointers) of one thread, and to start another thread by moving its context into the processor registers.
time slice	An equal portion of the processor's time.
Windows host	A Windows subsystem designated to provide INtime services (booting, configuration, file system proxy, etc.) to one or more RT clients. Not all Windows subsystems are Windows hosts.
Windows node	A computer executing a Windows subsystem.
Windows subsystem	A self-contained collection of software containing Windows, a number of Windows support processes, and the Windows portion of zero or more INtime applications.

Index

S A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A

accept *124*
AcceptRtControl *120*
alarms *26* , *32*
AllocateRtMemory *115*
allocating
    memory *46*
application development
    design *51*
    examples *54*
applications
    INscope *96*
    INtex *96*
    loading on RT node *79*
    RT Event Log Server *144*
    RT kernel loader *138*
    RT loader *137*
    RT Node Detection Server *147*
    RT Process Manager *146*
    sample, defined *9*
    SDM *96*
    Soft-Scope *96*
    Soft-Scope RT debugger *149*
    Windows NT Crash program *132*
AttachRtHeap *118*
AttachRtPort *118*

B

bind *124*
BindRtPort *118*
blue screen protection *12*
Blue.exe *132*
BlueCon.rta *132*
booting remote RT nodes
    from floppy disk *77*
browser, INtime RT Client *136*
BSOD. *See blue screen protection.*
bstring *124*
byteorder *124*

C

C and C++ sample for debugger *141*
calls
    NTX *109*
    real-time *110*
        low-level *110*
calls. *See system calls.*
CancelRtTransaction *119*
CancelTransaction *129*
CatalogRtHandle *117*
communicating
    between threads *44*
    between Windows NT and RT threads *4*
configuring INtime software
    default settings *65*
    interrupt resources *67*
    options *65*
connect *124*
ConnectRtPort *118*
conventions, notational *v*
conversion, records *148*
CopyRtSystemInfo *122*
crash program, Windows NT *132*
CreatePort *129*
CreateRtHeap *115*
CreateRtMailbox *113*
CreateRtMemoryHandle *115*
CreateRtPort *118*
CreateRtPortEx *118*
CreateRtRegion *120*
CreateRtSemaphore *121*
CreateRtThread *125*

D

data, validity of *46*
deadlock *35*
debug tools
    INscope *96*
    INtex *96*
    SDM *96*
    Soft-Scope *96*

debuggers 9
C and C++ sample 141
Soft-Scope 149
default configuration settings 65
DeletePort 129
DeleteRtHeap 115
DeleteRtMailbox 113
DeleteRtMemoryHandle 115
DeleteRtPort 118
DeleteRtRegion 120
DeleteRtSemaphore 121
DeleteRtThread 125
DeliverMessage 128
DeliverStatus 128
DeliverTransaction 128
DequeueInputTransaction 128
DequeueOutputTransaction 128
desigining RT applications
RT processes, appropriate tasks 53
sample system 54
Windows NT processes, assigning priority 53
designing RT applications
guidelines 51
DetachRtHeap 118
DetachRtPort 118
determinism 41
interrupt response time 41
thread switch time 41
developing INtime applications 6
development environment. *See INtime development environment.*
directories, object 31
DisableRtInterrupt 113
drivers
RT interface 19 , 144
RT USB interface 143
dynamic memory 30

E

e-mail address, TenAsys vi
email, technical support vi , 100
email, TenAsys v
Embedded C++ Help files 135
EnableRtInterrupt 113
encapsulating Windows NT
as an INtime software thread 19
See OSEM 18
encapsulation mechanism. *See OSEM.*
EnqueueInputTransactionTransaction 128
EnqueueOutPutTransaction 128
EnterRtInterrupt 112
EnterServiceRegion 129
Ethernet 17
event log service 144
event-driven applications 39
EventMsg.dll 133
events, alarms 26
examples
application systems 54
interrupt handlers 25 , 55
multitasking 55
mutual exclusion 34 , 35
regions 35
sample system 54
semaphores 34
synchronizing threads 45
exception handling calls 112
exception handling, floating point 141
exchange objects 32
mailboxes 33
ports 36
regions 35
semaphores 34
validation levels 32
exclusion, mutual 46
execution state 22
ExitRtProcess 120
ExitServiceRegion 129

F

Fault Handling (ntrobust) 141
file types
.DLL 7
.EXE 7
.RTA 7
files
required for RT nodes 75
Soft-Scope auxiliary 148
Finish 129
Floating Point Exception Handling 141
floppy disk
booting remote RT node from 77
FreeRtMemory 115
functions. *See system calls.*

G

GetAttributes *129*
GetFragment *129*
gethostname *124*
GetLastRtError *122*
getpeername *124*
GetPortId *128*
GetPortParameter *128*
GetRServiceAttributes *118*
GetRSVPTransaction *128*
GetRtBufferSize *116*
GetRtExceptionHandlerInfo *112*
GetRtHandleType *117*
GetRtHandleTypeEx *117*
GetRtHeapInfo *116*
GetRtInterruptLevel *112*
GetRtPhysicalAddress *115*
GetRtPortAttributes *118*
GetRtSize *115*
GetRtThreadAccounting *123* , *125*
GetRtThreadHandles *125*
GetRtThreadInfo *125*
GetRtThreadPriority *125*
GetRtThreadState *125*
getsockname *124*
getsockopt *124*
GetTransaction *128*
guidelines for designing applications *51*
guidelines for designing INtime applications *51*

H

HAL, modified for INtime software *11* , *21*
handlers
    interrupt *40*
    service *129*
handlers, interrupt
    alone *25*
    handler/thread combination *25*
header files
    NTX *139*
    RT *142*
help *iii* , *v* , *vi* , *3* , *100* , *133*

I

INBrow.ocx *136*
inet *124*
Initialize *129*
INscope application *96*
Installation program, running *60*
installing INtime software
    before you begin *60*
    running the Installation program *60*
InstallRtServiceDescriptor *118*
interface driver, RT *19*
interface, RT driver *144*
internal loop. *See loop.*
interrupt calls *112*
interrupt handlers *25* , *40*
    alone *25*
    examples *25* , *55*
    handler/thread combination *25*
interrupts
    during RT operation *19*
    levels *40*
    processing *40*
    resources, configuring *67*
    response time *41*
    threads *25* , *40*
INtex application *96*
INtime API Test *141*
INtime applications
    described *3*
    developing *6* , *51*
    files *7*
    how they run *13*
    loading on RT node at boot time *79*
    RT processes, appropriate tasks *53*
    runtime behavior *19*
    sample system *54*
    Windows NT processes, assigning priority *53*
INtime development environment
    debuggers *9*
    libraries *8*
    wizards *7*
INtime Driver Model Serial Driver Test Demo *141*
INtime Graphical Jitter *141*
INtime Performance Monitor.INtmPerf.*
        files.Performance monitor *136*
INtime RT Client Browser *136*
INtime RT device driver wizard *92*
INtime RT process add-in wizard *92*
INtime RT process wizard *90*
INtime runtime environment
    blue screen protection *12*

encapsulating Windows NT
as a thread 19
HAL, modified for Windows NT 11
HAL, modified for windows NT 21
how INtime applications run 13
kernel 11
memory protection 11
OSEM 11 , 18
RT interface driver 19
sample applications 9
INtime Serial Driver Sample 141
INtime services
server
INtime Registry service 148
INtime software
configuring 65
default settings 65
interrupt resources 67
debuggers 9
defined 3
libraries 8
requirements 59 , 75
wizards 7
INtime Software Overview Guide, PDF format 140
INtime USB Client sample 141
INtime Windows STOP Detection sample (STOPmgr) 141
INtime.hlp 133
INtimeConfiguration.exe 133
INtmCnfg.hlp 134
ip.rta 76
ItWrpSrv.exe 137
ITWrpSrv.hlp 134

J

Jitter 141

K

kernel
blue screen protection 12
loader 138
memory protection 11
RT 146
time management 26
knCreateRtAlarmEvent 126
knCreateRtMailbox 114
knCreateRtSemaphore 122
knDeleteRtAlarmEvent 126
knDeleteRtMailbox 114
knDeleteRtSemaphore 122
knGetKernelTime 126
knReleaseRtSemaphore 122
knResetRtAlarmEvent 126
knRtSleep 120
knSendRtData 114
knSendRtPriorityData 114
knSetKernelTime 126
knStartRtScheduler 120
knStopRtScheduler 120
knWaitForRtAlarmEvent 126
knWaitForRtData 114
knWaitForRtSemaphore 122

L

LdRta.exe 137
LdRta.hlp 134
levels, interrupt 40
libraries 8
NTX import 139
RT interface 142
limitations
maximum objects in system 27
listen 124
loader
RT application 137
RT kernel 138
loading INtime applications 79
LoadRtk.exe 138
LookupPortHandle 128
LookupRtHandle 117
loop. *See internal loop.*
low-level calls 110
when to use 110

M

mailbox calls 113
mailboxes, kernel 32 , 33 , 110
manager
RT process 146
managing time 26
MapRtPhysicalMemory 115
MapRtSharedMemory 115
memory
allocating and sharing 46

dynamic *30*
pools *30*
segments *30*
memory calls *114*
memory pools
defined *30*
memory protection *11*
memory/buffer management *114*
message port calls *117*
message transmission calls *119*
MFC*.dll files *138*
modified Windows NT Hardware Abstraction Layer. *See HAL.*
modular programming *38*
MsgBoxDemo (NTX Sample) *141*
multiprogramming *42*
multitasking, examples *55*
multi-threading *38*
mutual exclusion
defined *46*
examples *34* , *35*

N

ne.rta *75*
Node Detection Server *147*
nodes, RT
preparing for boot *75*
non-validating calls *110*
notational conventions *v*
NT Crash program *132*
ntrobust (Fault Handling) *141*
NTX calls *109*
NTX header files *139*
NTX import libraries *139*
NTX Sample (MsgBoxDemo) *141*
NTX.dll *139*
ntxAttachRtPort *117*
ntxBindRtPort *117*
ntxCancelRtTransaction *117*
ntxCatalogRtHandle *116*
ntxConnectRtPort *117*
ntxCopyRtData *115*
ntxCreateRtMailbox *113*
ntxCreateRtPort *117*
ntxCreateRtProcess *119*
ntxCreateRtSemaphore *121*
ntxDeleteRtMailbox *113*
ntxDeleteRtPort *117*
ntxDeleteRtSemaphore *121*
ntxDetachRtPort *117*
ntxFindINtimeNode *123*
ntxGetFirstLocation *123*
ntxGetLastRtError *122*
ntxGetLocationByName *123*
ntxGetNameOfLocation *123*
ntxGetNextLocation *123*
ntxGetRootRtProcess *116*
ntxGetRtErrorName *122*
ntxGetRtPortAttributes *117*
ntxGetRtServiceAttributes *117*
ntxGetRtSize *115*
ntxGetRtStatus *122*
ntxGetType *116*
ntxImportRthandle *116*
ntxLoadRtErrorString *122*
ntxLookupNtxhandle *116*
ntxMapRtSharedMemory *115*
ntxMapRtSharedMemoryEx *115*
ntxNotifyEvent *119*
ntxproxy.rta *75*
ntxReceiveData *113*
ntxReceiveHandle *113*
ntxReceiveRtMessage *117*
ntxReceiveRtReply *117*
ntxRegisterDependency *119*
ntxRegisterSponsor *119*
ntxReleaseRtBuffer *118*
ntxReleaseRtSemaphore *121*
NtxRemote2.exe *139*
ntxRequestRtBuffer *118*
ntxSendData *113*
ntxSendHandle *113*
ntxSendRtMessage *117*
ntxSendRtMessageRSVP *117*
ntxSetRtServiceAttributes *118*
ntxUncatalogRtHandle *116*
ntxUnmapRtSharedMemory *115*
ntxUnregisterDependency *119*
ntxUnregisterSponsor *119*
ntxWaitForRtSemaphore *121*

O

objects
defined *27*

directories *31* , *116*
exchange *32*
mailboxes *33*
ports *36*
regions *35*
semaphores *34*
maximum in system *27*
memory pools *30*
processes *28*
threads *28*
OS encapsulation mechanism. *See OSEM.*
OSEM *11* , *18*
OvwGuide.pdf *140*

P

PCI library calls *130*
PciClassName *130*
PciDeviceName *130*
PciFindDevice *130*
PciGetConfigRegister *130*
PciInitialize *130*
PciReadHeader *130*
PciSetConfigRegister *130*
PciVendorName *130*
PciClassName *130*
PciDeviceName *130*
PciFindDevice *130*
PciGetConfigRegister *130*
PciInitialize *130*
PciReadHeader *130*
PciSetConfigRegister *130*
PciVendorName *130*
PDF file, *INtime Software Overview Guide* *140*
Pds.* files *140*
Ping.rta *140*
ping.rta *75*
pools, memory *30*
port calls
message transmission *119*
port object management *118*
service support *118*
port callst *117*
port object management calls *118*
ports *36*
pre-emptive scheduling *39*
priority-based scheduling *22* , *39*
process management calls *119*
processes *28*
about *28*
components *42*
manager, RT *146*
tree *29*
processing, interrupts *40*
Project files *140*
protection
blue screen *12*
memory *11*

Q

QueryInputTransactionQueue *128*
QueryOutputTransactionQueue *128*

R

real-time
debuggers *9*
interface driver *19*
INtime applications, designing *51*
response *37*
RT kernel *11*
real-time calls *110*
Real-time Interrupt Sample *141*
ReceiveRtData *114*
ReceiveRtFragment *119*
ReceiveRtHandle *114*
ReceiveRtMessage *119*
ReceiveRtReply *119*
record conversion *148*
recursive. *See recursive.*
recv *124*
region calls *120*
regions *35*
RegisterRtDependency *120*
RegisterRtSponsor *120*
registry calls *127*
ReleaseControlBuffer *128*
ReleaseRtBuffer *116*
ReleaseRtControl *120*
ReleaseRtSemaphore *121*
ReleaseTransaction *128*
remote RT node
booting from floppy disk *77*
ReportRtEvent *122*
RequestControlBuffer *128*
RequestRtBuffer *116*

RequestTransaction *128*
requirements for INtime software *59* , *75*
ResetRtInterruptHandler *112*
response time, predictability *41*
ResumeRtThread *125*
rip.rta *76*
round-robin scheduling *24*
rrtimage *75*
rrtimage.dbg *75*
RS-232 *17*
RSL Example *141*
RT application loader *137*
RT Clock Synchronization Server *143*
RT debugger
    Soft-Scope *149*
RT Device Driver wizard *143*
RT Event Log Server *144*
RT header files *142*
RT I/O console *145*
RT I/O Server *145*
RT interface driver *144*
RT interface libraries *142*
RT kernel *11* , *146*
    blue screen protection *12*
    communicating with Windows NT *4*
    loader *138*
RT node
    files *141*
    preparing for boot *75*
    required files *75*
RT Node Detection Server *147*
RT Process Add-in wizard *147*
RT Process Manager *146*
RT process wizard *147*
RT service handlers *129*
    CancelTransaction *129*
    CreatePort *129*
    DeletePort *129*
    Finish *129*
    GetAttributes *129*
    GetFragment *129*
    Initialize *129*
    SendMessage *130*
    Service *130*
    SetAttributes *130*
    UpdateReceiveInfo *130*
    VerifyAddress *130*
RT services *111*
RT Shared Library wizard *148*
RT USB Interface Drivers *143*
RtAddinW.hlp *134*
rtcimcom.rta *76*
rtcimudp.rta *76*
RtClkSrv *143*
RtDrvrW.hlp *134*
RtDrvrW5.awx *143*
RtELServ.exe *144*
RTIF.SYS *17* , *19*
RtIf.sys *144*
RtIOCons.exe *145*
RtIOSrv.exe *145*
RtkImage.dbg *146*
RtLoad.sys file *80*
RtMgr.exe *146*
RTMgr.hlp *134*
RtNdSrv.exe *147*
RtPrcMgr.rta *147*
RtProc5.awx *147*
RtProcAddinW5.awx *147*
RtProcW.hlp *134*
RtRegCloseKey *127*
RtRegConnectRegistry *127*
RtRegCreateKeyEx *127*
RtRegDeleteKey *127*
RtRegDeleteValue *127*
RtRegEnumKeyEx *127*
RtRegEnumValue *127*
RtRegFlushKey *127*
RtRegLoadKey *127*
RtRegOpenKeyEx *127*
RtRegQueryInfoKey *127*
RtRegQueryValueEx *127*
RtRegReplaceKey *127*
RtRegRestoreKey *127*
RtRegSaveKey *127*
RtRegSetValueEx *127*
RtRegSrv.exe *148*
RtRegUnLoadKey *127*
RtRslWiz.awx *148*
rtsetup.exe *76*
RtSleep *125*
runtime environment. *See INtime runtime environment.*

S

sample application folders *140*
sample applications *9*
scheduler calls *120*
scheduling
  priority-based *22* , *39*
  round-robin *24*
SDM application *96*
segments *30*
semaphores *34* , *121*
send *124*
SendMessage *130*
SendRtData *114*
SendRtHandle *113*
SendRtMessage *119*
SendRtMessageRSVP *119*
SendRtReply *119*
serdrvr.rta *76*
Serial I/O *17*
Service *130*
service handlers *129*
service support calls *118*
Service wrapper *137*
Services
  RT Event Log Server *144*
  RT I/O Server *145*
  RT Node Detection Server *147*
services, RT *111*
ServicesLRT Clock Synchronization Server *143*
SetAttributes *130*
SetLastRtError *122*
SetPortParameter *128*
SetRtExceptionHandler *112*
SetRtInterruptHandler *112*
SetRtInterruptHandlerEx *112*
SetRtProcessMaxPriority *125*
SetRTServiceAttributes *118*
SetRtSystemAccountingMode *123*
SetRtThreadPriority *125*
sharing memory *46*
shutdown *124*
SignalEndOfRtInterrupt *113*
SignalRtInterruptThread *113*
socket *124*
Soft-Scope
  application *96*
  auxiliary files *148*
  debugger *9*
  record conversion *148*
  RT debugger *149*
SSBug.exe *148*
SSWin32.exe *149*
Starvation Alarm *150*
StarveAlarm.exe *150*
states
  execution *22*
  thread *41*
STOPmgr (INtime Windows STOP Detection sample) *141*
StrvAlrm.hlp *134*
support *iii* , *v* , *3* , *133*
support, technical *vi* , *100*
SuspendRtThread *125*
switches
  RtLoad.sys file *80*
synchronizing threads *45*
system accounting calls *123*
system calls *8* , *48*
  high-level calls
    exception handling *112*
    interrupts *112*
    mailboxes *113*
    memory *114*
    message ports *117*
    object directories *116*
    process management *119*
    regions *120*
    semaphores *121*
    system accounting *123*
    thread management *125*
  low-level calls *121*
    mailboxes *113*
    scheduler *120*
  NTX calls *109*
    mailboxes *113*
    memory *114*
    object directories *116*
    semaphores *121*
  ntx calls
    system data *123*
  PCI library *130*
  real-time *110*
    low-level *110*
  registry *127*

TCP/IP calls 124
types 108
system data calls 123

T

TCP Client 141
TCP Server 141
tcp.rta 76
TCP/IP calls 124
TCPClient 141
TCPServ 141
technical support vi , 100 , 133
TenAsys, contacting vi
thread management calls 125
threads 28
about 28
communicating between Windows NT and the RT kernel 4
communication 43
coordination 43
encapsulating Windows NT 19
execution state 22
interrupt 40
message passing 44
multi-threading 38
priority of 22
scheduling
defined 22
priority-based 22
round-robin 24
states 41
switch time 41
synchronizing 45
time, managing 26
top-down programming 38
troubleshooting iii , v , 3 , 133

U

udp.rta 76
UDP/IP 17
UncatalogRtHandle 117
UninstallRtServiceDescriptor 118
UnregisterRtDependency 120
UnregisterRtSponsor 120
UpdateReceiveInfo 130
USB Client sample 141
Utilities
INtime Configuration utility 133

V

validation levels 32
VerifyAddress 130

W

WaitForRtControl 120
WaitForRtInterrupt 113
WaitForRtSemaphore 121
Windows NT
communicating with the RT kernel 4
encapsulated as an INtime software thread 19
modified HAL 21
running INtime applications 3 , 13
where to find information vi
wizards 7 , 90 , 92
RT Device Driver 143
RT Process Add-in, MSVC 5.0 147
RT process, MSVC 5.0 147
RT Shared Library 148